RYDE Signature Scheme

02/06/2025

Submitters (alphabetical order):

- Nicolas Aragon (Naquidis Center, FR)
- Magali Bardet (LITIS, University of Rouen Normandie & INRIA, FR)
- Loïc Bidoux (Technology Innovation Institute, UAE)
- Jesús-Javier Chi-Domínguez (Technology Innovation Institute, UAE)
- Victor Dyseryn (Télécom Paris, FR)
- Thibauld Feneuil (Cryptoexperts, FR)
- Philippe Gaborit (University of Limoges, FR)
- Antoine Joux (CISPA, Helmholtz Center for Information Security, DE)
- Romaric Neveu (University of Limoges, FR)
- Matthieu Rivain (Cryptoexperts, FR)
- Jean-Pierre Tillich (INRIA, FR)
- Adrien Vincotte (University of Limoges, FR)

Contact: team@pqc-ryde.org

Version: 2.0.1

Table of Contents

List of Figures

List of Tables

List of Algorithms

Changelog

Version 2.0.1 (02/06/2025)

- The implementations of RYDE have been improved (performance speed-up, fixing constant-time issues, removing dynamic allocation...).

Version 2.0.0 (05/02/2025)

- The design of RYDE have been improved and now relies on the TCitH framework [FR23b,FR23a] (or alternatively the VOLEitH framework) along with the Dual Support Modeling [BFG+24]. As a result, RYDE signature sizes have been significantly reduced.

- Romaric Neveu have joined the RYDE team.

1 Introduction

RYDE is a post-quantum signature digital signature scheme based on the hardness of the Rank Syndrome Decoding (RSD). Informally, given a syndrome $\boldsymbol{y}$ and a parity-check matrix $\boldsymbol{H}$, the Rank Syndrome Decoding problem asks to find an error $\boldsymbol{x}$ of small rank weight, whose syndrome is $\boldsymbol{y}$. RYDE relies on a Zero-Knowledge Proof of Knowledge (ZKPoK) of an RSD solution. This ZKPoK is based on the Multi Party Computation in the Head (MPCitH) paradigm [IKOS07]. In particular, RYDE relies on the Threshold Computation in the Head (TCitH) framework [FR23b,FR23a]. A variant using VOLE in the Head (VOLEitH) framework [BBD+23] is also described. The ZKPoK is then converted into a signature scheme using the Fiat-Shamir transform [FS87].

This document is structured as follows. We present in Section 2 the mathematical background and the notations we will use. Then, Sections 3 and 4 deal with the high-level description and the detailed algorithmic description of the scheme respectively. Sections 5 and 6 are dedicated to the parameters and the performances of RYDE. A security analysis of RYDE is provided in Sections 7 and best known attacks are presented in Section 8. Finally, in Section 9, we summarize the main advantages and limitations of RYDE.

2 Mathematical Background and Notation

2.1 Notation

We summarize the main mathematical notations in Table 1. We employ lower-case and upper-case bold letters for rows vectors and matrices, respectively.

Symbol	Meaning
$\{0,1\}^\ell$	Set of binary strings of length ℓ.
$\{0,1\}^*$	Set of binary strings of finite length.
$\mathbb{F}_q$	The finite field of q elements.
$\mathbb{F}_q^{n_c \times n_r}$	The vector space of $n_c \times n_r$ matrices over the field $\mathbb{F}_q$.
$\mathbf{0}_{n_c \times n_r}$	The $n_c \times n_r$ zero matrix.
$\boldsymbol{I}_s$	The $s \times s$ identity matrix.
$\boldsymbol{M}^\top$	The transposed matrix of $\boldsymbol{M}$
$\mathsf{rank}(\boldsymbol{M})$	The rank of the matrix $\boldsymbol{M}$.
$(\boldsymbol{A} \mid \boldsymbol{B})$	The matrix obtained by juxtaposing the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.
log	The logarithm in base 2.
$\otimes$	Bitwise multiplication (AND).
$\oplus$	Bitwise addition (XOR).

Table 1. Mathematical notation.

2.2 The Rank Syndrome Decoding problem

It is first fitting to recall some background about the Rank Metric.

Definition 1 (Rank Metric over $\mathbb{F}_{q^m}^n$). *Let $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_{q^m}^n$, and $\mathcal{B} = (b_1, \ldots, b_m) \in \mathbb{F}_{q^m}^m$ an $\mathbb{F}_q$-basis of $\mathbb{F}_{q^m}$. Each coordinate x_j can be associated with a vector $(x_{j,1}, \ldots, x_{j,m}) \in \mathbb{F}_q^m$ such that $x_j = \sum_{i=1}^m x_{j,i} b_i$. Let us define the following notations:*

- $\boldsymbol{M}_{\boldsymbol{x}} = (x_{j,i})_{(j,i) \in [1,n] \times [1,m]}$ *is the* matrix associated to the vector $\boldsymbol{x}$*;*
- *the* rank weight *is defined as:* $w_R(\boldsymbol{x}) = \mathsf{rank}(\boldsymbol{M}_{\boldsymbol{x}})$*;*
- *the* distance *between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{F}_{q^m}^n$ is:* $d(x,y) = w_R(\boldsymbol{x} - \boldsymbol{y})$*;*
- *the* support *of a vector* $\mathrm{Supp}(\boldsymbol{x})$ *is the $\mathbb{F}_q$-linear subspace of $\mathbb{F}_{q^m}$ generated by its coordinates:* $\mathrm{Supp}(\boldsymbol{x}) = \langle x_1, \ldots, x_n \rangle$.

Definition 2. *A linear code $\mathcal{C}$ over $\mathbb{F}_{q^m}$ of dimension k and length n is a linear subspace of $\mathbb{F}_{q^m}^n$ of dimension k. The elements of $\mathcal{C}$ are called codewords. The code $\mathcal{C}$ can be represented in two ways:*

- *by a generator matrix* $\boldsymbol{G}$*, where* $\mathcal{C} = \{\boldsymbol{mG}, \boldsymbol{m} \in \mathbb{F}_{q^m}^k\}$*, or*
- *by a parity-check matrix* $\boldsymbol{H} \in \mathbb{F}_{q^m}^{(n-k)\times n}$ *where* $\mathcal{C} = \{\boldsymbol{x} \in \mathbb{F}_{q^m}^n : \boldsymbol{H}\boldsymbol{x}^\top = \boldsymbol{0}^\top\}$

We now continue by formally recalling the definition of the rank syndrome decoding (RSD) problem.

Definition 3 (RSD problem). *Let* q*,* m*,* n*,* k *and* r *be positive integers. Let* $\boldsymbol{H} \xleftarrow{\$} \mathbb{F}_{q^m}^{(n-k)\times n}$ *and* $\boldsymbol{x} \xleftarrow{\$} \mathbb{F}_{q^m}^n$ *such that* $w_R(\boldsymbol{x}) = r$*. Let* $\boldsymbol{y}^\top = \boldsymbol{H}\boldsymbol{x}^\top$*. Given* $(\boldsymbol{H}, \boldsymbol{y})$*, the computational* $\mathsf{RSD}(q, m, n, k, r)$ *problem asks to find a vector* $\tilde{\boldsymbol{x}} \in \mathbb{F}_{q^m}^n$ *such that* $\boldsymbol{H}\tilde{\boldsymbol{x}}^\top = \boldsymbol{y}^\top$ *and* $w_R(\tilde{\boldsymbol{x}}) = r$*.*

The RYDE signature schemes relies on a variant of the RSD problem, namely, the $\mathsf{RSD}_{\mathrm{s}}$ variant. This variant reduces in polynomial-time to RSD [BFG+24].

Definition 4 ($\mathsf{RSD}_{\mathbf{s}}$ problem). *Let* q*,* m*,* n*,* k *and* r *be positive integers. Let* $\boldsymbol{H} \xleftarrow{\$} \mathbb{F}_{q^m}^{(n-k)\times n}$ *and* $\boldsymbol{x} = (x_i) \xleftarrow{\$} \mathbb{F}_{q^m}^n$ *such that* $w_R(\boldsymbol{x}) = r$*,* $x_1 = 1 \in \mathbb{F}_{q^m}$ *and* $\langle x_1, \ldots, x_r\rangle_{\mathbb{F}_q} = \mathrm{Supp}(\boldsymbol{x})$*. Let* $\boldsymbol{y}^\top = \boldsymbol{H}\boldsymbol{x}^\top$*. Given* $(\boldsymbol{H}, \boldsymbol{y})$*, the computational* $\mathsf{RSD}_s(q, m, n, k, r)$ *problem asks to find a vector* $\tilde{\boldsymbol{x}} \in \mathbb{F}_{q^m}^n$ *such that* $\boldsymbol{H}\tilde{\boldsymbol{x}}^\top = \boldsymbol{y}^\top$ *and* $w_R(\tilde{\boldsymbol{x}}) = r$*.*

3 High-Level Description of RYDE

In this section, we provide a high-level description of the RYDE signature scheme. This scheme is built by applying the Fiat-Shamir transformation on top of a zero-knowledge proof of knowledge for a solution to a Rank Syndrome Decoding instance. The underlying proof system uses the TCitH [FR23b,FR23a] framework while parameters using the VOLEitH [BBD+23] framework are given in Appendix A.

3.1 TCitH Framework in the PIOP Formalism

The MPCitH paradigm [IKOS07] is a versatile method introduced in 2007 to build zero-knowledge proof systems using techniques from secure multi-party computation (MPC). This paradigm has been drastically improved in recent years and is particularly efficient to build zero-knowledge proofs for small circuits such as those involved in (post-quantum) signature schemes. The more recent MPCitH-based frameworks are the VOLE-in-the-Head (VOLEitH) framework from [BBD+23] and the Threshold-Computation-in-the-Head (TCitH) framework from [FR23b,FR23a].

In this subsection, we will describe the general proof system on which RYDE is relying on. In what follows, we present this proof system using the formalism of the Polynomial Interactive Oracle Proofs (PIOP), as presented in [Fen24]. Indeed, while the TCitH and VOLEitH frameworks have been respectively introduced using a sharing-based and a VOLE-based formalism, one can unify those two frameworks using the PIOP formalism, which enables us to have a description that does not depend on MPC technologies[1], leading to an easier-to-understand scheme for those who do not already know those two frameworks.

Let us assume that we want to build an interactive zero-knowledge proof that would enable a prover to convince a verifier that he knows a witness $w \in \mathbb{F}_q^n$ which satisfies some public polynomial relations:

$$\text{for all } 1 \leq j \leq m,\ f_j(w) = 0,$$

where $f_1, \ldots, f_m$ are polynomials over $\mathbb{F}_q$ of total degree at most d. Let us consider two proof parameters $N, \mu \in \mathbb{N}$ such that $N \leq 2^\mu$. The proof system we consider is the following:

1. For all $1 \leq j \leq n$, the prover samples a random degree-1 polynomials P_j such that $P_j(X) = w_j \cdot X + (w_{\mathsf{base}})_j$ for some $(w_{\mathsf{base}})_j \in \mathbb{F}_{q^\mu}$. He also samples a random degree-$(d-1)$ polynomial $P_0 \in \mathbb{F}_{q^\mu}[X]$. He commits to those polynomials.

[1] In the TCitH framework, instead of performing operations over Shamir's secret sharings, we can directly work over their underlying polynomials. In the VOLEitH framework, instead of performing operations over VOLE gadgets, we can directly work over their underlying degree-1 polynomial.

2. The verifier chooses random coefficients $\gamma_1, \ldots, \gamma_m$ from $\mathbb{F}_{q^\mu}$ and sends them to the prover. The latter then reveals the degree-$(d-1)$ polynomial $Q(X)$ defined as

$$Q(X) = P_0(X) + \sum_{j=1}^{m} \gamma_j \cdot f_j^{[h]}(X, P_1(X), \ldots, P_n(X)), \tag{1}$$

   where $f_j^{[h]}$ is the homogeneous version of the polynomial f_j, *i.e.*

$$f_j^{[h]}(Y, X_1, \ldots, X_n) := Y^d \cdot f_j(\frac{X_1}{Y}, \ldots, \frac{X_n}{Y}) \ .$$

3. The verifier samples a random evaluation point r from a public subset $S \subset \mathbb{F}_{q^\mu}$ of size N and sends it to the prover. The latter then reveals the evaluations $v_i := P_i(r)$, together with a proof π that the evaluations are consistent with the commitment.

4. The verifier checks that the revealed evaluations are consistent with the commitment using π and checks that we have

$$Q(r) = v_0 + \sum_{j=1}^{m} \gamma_j \cdot f_j^{[h]}(r, v_1, \ldots, v_n) \ . \tag{2}$$

The above protocol assumes that the prover has a way to commit polynomials and to provably open some evaluations later (while keeping hidden the other evaluations).

Security Analysis. We can observe that the coefficient in front of the degree-d monomial in the right term of Equation (1) is

$$\sum_{j=1}^{m} \gamma_j \cdot f_j(w_1, \ldots, w_n) \ , \tag{3}$$

so the degree-$(d-1)$ polynomial Q is well-defined because this quantity is zero when the witness w is valid. Let us assume that the prover is malicious, meaning that he does not a valid witness. It implies that there exists j^* such that $f_{j^*}(w) \neq 0$. In that case, the probability that there exists some Q such that Equation (1) holds is at most $1/q^\mu$ over the randomness of $\gamma_1, \ldots, \gamma_m$, because the coefficient (3) is zero only with probability $1/q^\mu$. If Equation (1) does not hold, the probability that the check in Equation 2 passes is at most d/N, since the degree-d polynomial relation

$$Q(X) - \left(P_0 + \sum_{j=1}^{m} \gamma_j \cdot f_j^{[h]}(X, P_1(X), \ldots, P_m(X)) \right) \neq 0$$

would have at most d roots (and so the random challenge r should be among those roots). So, the proof system is *sound*, with a soundness error of $\frac{1}{q^\mu} + \left(1 - \frac{1}{q^\mu}\right) \cdot \frac{d}{N}$. Moreover, assuming that the commitment scheme is hiding, we can observe that the interactive proof is zero-knowledge since

- revealing $Q(X)$ leaks no information about the secret thanks to the random polynomial P_0, and
- revealing one evaluation of the polynomials $P_1, \ldots, P_n$ leaks no information about the leading term thanks to the randomness used to build those polynomials.

In our setting of the RYDE signature scheme, $\mathbb{F}_{q^\mu}$ might be a small field, so it would lead to a relatively bad soundness error. To solve this issue, we just repeat Step 2 of the proof system ρ times in parallel: the verifier chooses a random matrix $\boldsymbol{\Gamma} \in \mathbb{F}^{\rho \times m}$ and then the prover reveals ρ polynomials $Q_1, \ldots, Q_\rho$ such that

$$Q_k(X) = P_{0,k}(X) + \sum_{j=1}^{m} \boldsymbol{\Gamma}_{k,j} \cdot f_j^{[h]}(X, P_1(X), \ldots, P_n(X))$$

where $P_{0,1}, \ldots, P_{0,\rho}$ are ρ random degree-$(d-1)$ polynomials that have been committed in the previous step. In that case, the soundness error is now $\frac{1}{q^{\mu \cdot \rho}} + \left(1 - \frac{1}{q^{\mu \cdot \rho}}\right) \cdot \frac{d}{N}$.

Remark 1. Using the above tweak, when taking ρ such that $\rho \cdot \mu \cdot \log_2 q \geq \lambda$, the soundness error is roughly d/N. When N is small (compared to 2^λ), we would need to repeat the zero-knowledge protocol several times to achieve the desired security level. One solution could be that the verifier checks the polynomial relation (1) into several points instead of a single one. If the verifier checks the relation into ℓ points, the prover needs to sample $P_1, \ldots, P_n$ as degree-ℓ polynomials to preserve zero-knowledge, and the soundness error would be $\frac{1}{q^{\mu \cdot \rho}} + \left(1 - \frac{1}{q^{\mu \cdot \rho}}\right) \cdot \frac{\binom{d \cdot \ell}{\ell}}{\binom{N}{\ell}}$. By taking ℓ such the soundness error is directly negligible, we would not need to rely on parallel repetitions. However, the techniques we would like to use to commit to polynomials are based on GGM trees and do not scale well if we want to open several evaluations. We would need to use techniques based on Merkle trees (*e.g.* TCitH-MT [FR23a]), but the signature size would be larger than 6 KB (for the first security level).

In what follows, we describe how to commit polynomials such that we can later open some evaluations.

The TCitH-GGM Approach. The TCitH framework [FR23a] shows that we can commit $\bar{n}$ random polynomials using seed trees thanks to ideas from [ISN89,CDI05]. Here is the commitment process for degree-1 polynomials:

1. One uses an all-but-one vector commitment (AVC) to sample and commit N seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_N$.
2. One expands each seed_i as $w_{\mathsf{rnd},i} := \mathsf{PRG}(\mathsf{seed}_i) \in \mathbb{F}_q^{\bar{n}}$ for $i \in \{1, \ldots, N\}$, where PRG is a pseudorandom generator.

3. One computes
$$w_{\mathsf{acc}} \leftarrow \sum_{i=1}^{N} w_{\mathsf{rnd},i} \in \mathbb{F}_q^{\bar{n}}$$
$$w_{\mathsf{base}} \leftarrow -\sum_{i=1}^{N} \phi(i) \cdot w_{\mathsf{rnd},i} \in \mathbb{F}_{q^\mu}^{\bar{n}}$$
where $\phi : \{1, \ldots, N\} \to \mathbb{F}_{q^\mu}$ is a public one-to-one function.
4. One reveals the auxiliary value $\mathsf{aux} := w - w_{\mathsf{acc}}$.
5. One defines P_j as
$$P_j(X) = w_j \cdot X + (w_{\mathsf{base}})_j$$
for all j.

This commitment procedure has the main advantage to enable the prover to reveal one evaluation $\{P_j(\phi(i^*))\}_j$ for $i^* \in [1 : N]$ while keeping *secret* the coefficient w and w_{base}: they just need to reveal all the $\{\mathsf{seed}_i\}_i$ except seed_{i^*} (by opening the AVC scheme) and the verifier will be able to compute $P_j(\phi(i^*))$ as

$$\phi(i^*) \cdot \mathsf{aux}_j + \sum_{i=1, i \neq i^*}^{N} (\phi(i^*) - \phi(i)) \cdot (w_{\mathsf{rnd},i})_j \quad \text{with} \quad w_{\mathsf{rnd},i} := \mathsf{PRG}(\mathsf{seed}_i).$$

Indeed, we have that

$$\begin{aligned}
\phi(i^*) \cdot \mathsf{aux}_j + & \sum_{i=1, i \neq i^*}^{N} (\phi(i^*) - \phi(i)) \cdot (w_{\mathsf{rnd},i})_j \\
&= \phi(i^*) \cdot \left(\mathsf{aux}_j + \sum_{i=1}^{N} (w_{\mathsf{rnd},i})_j \right) - \sum_{i=1}^{N} \phi(i) \cdot (w_{\mathsf{rnd},i})_j \\
&= \phi(i^*) \cdot (\mathsf{aux}_j + (w_{\mathsf{acc}})_j) + (w_{\mathsf{base}})_j \\
&= \phi(i^*) \cdot w_j + (w_{\mathsf{base}})_j = P_j(\phi(i^*))
\end{aligned}$$

Using this commitment procedure, the zero-knowledge protocol has 5 rounds, and one needs to rely on protocol repetitions to achieve the desired security. Indeed, the computational complexity is linear in N and so we can not take N exponentially large. To have a λ-bit security we need to repeat the protocol τ times in parallel, such that $(d/N)^\tau \leq 2^{-\lambda}$, assuming that $q^{-\mu \cdot \rho}$ is negligible.

The VOLEitH Approach. As the TCitH framework, the VOLEitH approach starts by committing τ sets of polynomials $\{P_i^{(1)}\}_i, \ldots, \{P_i^{(\tau)}\}_i$ in parallel. However, instead of considering those sets of polynomials individually as the TCitH framework, the VOLEitH framework [BBD+23] consists in "merging them" into polynomials for which we will be able to open N^τ evaluations. This merge introduces an additional round in the proof system, leading to a 7-round proof system with soundness error

$$\frac{1}{2^{\mu \cdot \rho}} + \left(1 - \frac{1}{2^{\mu \cdot \rho}}\right) \cdot \frac{d}{N^\tau} \ .$$

3.2 Dual Support Modeling

As explained previously, the RYDE signature scheme is built from a zero-knowledge proof of knowledge for a solution to an RSD_s instance. We rely on the proof system described in the previous section. This scheme enables us to prove the knowledge of a witness satisfying some degree-d polynomial constraints, so we should write the RSD_s problem as a system of degree-d polynomial equations.

Instead of q-polynomials, RYDE uses the Dual Support Modeling from [BFG+24]. In this setting, the protocol aims at verifying that a vector $\boldsymbol{x}$ is solution of the constraints

$$\begin{cases} \boldsymbol{x}\boldsymbol{H}^\top = \boldsymbol{y}, \\ w_R(\boldsymbol{x}) \leq r \end{cases}$$

for a given matrix $\boldsymbol{H} \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and a given vector $\boldsymbol{y} \in \mathbb{F}_q^{n-k}$. The modeling consists in viewing $\boldsymbol{x}$ as a vector-matrix product, $\boldsymbol{x} = \boldsymbol{s}\boldsymbol{C}$, with $\boldsymbol{s} \in \mathbb{F}_{q^m}^r$ and $\boldsymbol{C} \in \mathbb{F}_q^{r\times n}$. Furthermore, the modeling specializes the matrix $\boldsymbol{C}$ as $\begin{bmatrix}\boldsymbol{I}_r & \boldsymbol{C'}\end{bmatrix}$ for some matrix $\boldsymbol{C'} \in \mathbb{F}_q^{r\times(n-r)}$, and the vector $\boldsymbol{s}$ as $(1 \parallel \boldsymbol{s'})$ for some vector $\boldsymbol{s'} \in \mathbb{F}_{q^m}^{r-1}$. Therefore, we use the proof system to prove that we know the witness $(\boldsymbol{s'}, \boldsymbol{C'})$ which satisfy the following *quadratic* constraints ($d = 2$):

$$\boldsymbol{s}\boldsymbol{C}\boldsymbol{H}^\top = \boldsymbol{y} \quad \text{with} \quad \boldsymbol{C} = \begin{bmatrix}\boldsymbol{I}_r & \boldsymbol{C'}\end{bmatrix} \quad \text{and} \quad \boldsymbol{s} = (1 \parallel \boldsymbol{s'}).$$

3.3 RYDE Protocol

Let us now express the proof system in the specific case of the RYDE scheme, *i.e.* specialize the proof system described in Section 3.1 for the Rank Syndrome Decoding modeling of Section 3.2:

1. The prover begins by sampling random degree-1 polynomials $P_{s'} = \boldsymbol{s'} \cdot X + \boldsymbol{s'}_{\mathsf{base}}$ and $P_{C'} = \boldsymbol{C'} \cdot X + \boldsymbol{C'}_{\mathsf{base}}$, where $\boldsymbol{s'}_{\mathsf{base}}$ and $\boldsymbol{C'}_{\mathsf{base}}$ are randomly sampled from $\mathbb{F}_{q^m}^{r-1}$ and $\mathbb{F}_q^{r\times(n-r)}$ respectively. He also samples a random degree-1 polynomial $P_v = \boldsymbol{v} \cdot X + \boldsymbol{v}_{\mathsf{base}}$ where $\boldsymbol{v} \in \mathbb{F}_{q^m}^\rho$ and $\boldsymbol{v}_{\mathsf{base}} \in \mathbb{F}_{q^m}^\rho$. He commits to those polynomials.
2. The verifier chooses a random matrix $\boldsymbol{\Gamma} \in \mathbb{F}_{q^m}^{(n-k)\times\rho}$ and sends it to the prover. The latter reveals the degree-1 polynomial $P_\alpha(X)$ defined as

$$P_\alpha(X) = P_v(X) + \boldsymbol{\Gamma} \cdot \left(\boldsymbol{H} \cdot P_x(X) - \boldsymbol{y} \cdot X^2\right) \in (\mathbb{F}_{q^m}[X])^\rho$$

   with

$$P_x(X) = (X \parallel P_{s'}(X)) \cdot [P_{I_r}(X) \parallel P_{C'}] \in (\mathbb{F}_{q^m}[X])^n \ ,$$

   where $P_{I_r} := \boldsymbol{I}_r \cdot X \in (\mathbb{F}_q[X])^{r\times r}$.
3. The verifier samples a random evaluation point r from a public subset $S \subset \mathbb{F}_{q^m}$ of size N and sends it to the prover. The latter then reveals the evaluations $\boldsymbol{s'}_{\mathsf{Eval}} := P_{s'}(r)$, $\boldsymbol{C'}_{\mathsf{Eval}} := P_{C'}(r)$ and $\boldsymbol{v}_{\mathsf{Eval}} := P_v(r)$, together with a proof π that the evaluations are consistent with the commitment.

4. The verifier checks that the revealed evaluations are consistent with the commitment using π and check that we have

$$P_\alpha(r) =_? \boldsymbol{v}_{\mathsf{Eval}} + \boldsymbol{\Gamma} \cdot \left(\boldsymbol{x}_{\mathsf{Eval}} \cdot \boldsymbol{H}^\top - \boldsymbol{y} \cdot r^2\right) \in \mathbb{F}_{q^m}^{\rho}$$

with

$$\boldsymbol{x}_{\mathsf{Eval}} = (r \parallel \boldsymbol{s'}_{\mathsf{Eval}}) \cdot [r \cdot \boldsymbol{I}_r \parallel \boldsymbol{C'}_{\mathsf{Eval}}].$$

Committing to Witness Polynomials. In the above proof system, we need to commit three polynomials:

- The witness polynomial $P_{s'} = \boldsymbol{s'} \cdot X + \boldsymbol{s'}_{\mathsf{base}}$ encoding the secret $\boldsymbol{s'}$;
- The witness polynomial $P_{C'} = \boldsymbol{C'} \cdot X + \boldsymbol{C'}_{\mathsf{base}}$ encoding the secret $\boldsymbol{C'}$;
- The masking polynomial $P_v = \boldsymbol{v} \cdot X + \boldsymbol{v}_{\mathsf{base}}$, which aims to avoid leakage through P_α.

To commit them, we use the TCitH approach as explained in Section 3.1:

1. We use an all-but-one vector commitment (AVC) to sample and commit N seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_N$.
2. One expands each seed_i as

$$(\boldsymbol{s'}_{\mathsf{rnd},i}, \boldsymbol{C'}_{\mathsf{rnd},i}, \boldsymbol{v}_{\mathsf{rnd},i}) := \mathsf{PRG}(\mathsf{seed}_i) \in \mathbb{F}_{q^m}^{r-1} \times \mathbb{F}_q^{r\times(n-r)} \times \mathbb{F}_{q^m}^{\rho\times 1}$$

   for all $i \in \{1, \ldots, N\}$.
3. One computes

$$(\boldsymbol{s'}_{\mathsf{acc}}, \boldsymbol{C'}_{\mathsf{acc}}, \boldsymbol{v}_{\mathsf{acc}}) \leftarrow \left(\sum_{i=1}^{N} \boldsymbol{s'}_{\mathsf{rnd},i},\ \sum_{i=1}^{N} \boldsymbol{C'}_{\mathsf{rnd},i},\ \sum_{i=1}^{N} \boldsymbol{v}_{\mathsf{rnd},i}\right)$$
$$(\boldsymbol{s'}_{\mathsf{base}}, \boldsymbol{C'}_{\mathsf{base}}, \boldsymbol{v}_{\mathsf{base}}) \leftarrow \left(-\sum_{i=1}^{N} \phi(i) \cdot \boldsymbol{s'}_{\mathsf{rnd},i},\ -\sum_{i=1}^{N} \phi(i) \cdot \boldsymbol{C'}_{\mathsf{rnd},i},\ -\sum_{i=1}^{N} \phi(i) \cdot \boldsymbol{v}_{\mathsf{rnd},i}\right)$$

   with $(\boldsymbol{s'}_{\mathsf{acc}}, \boldsymbol{C'}_{\mathsf{acc}}, \boldsymbol{v}_{\mathsf{acc}}) \in \mathbb{F}_{q^m}^{r-1} \times \mathbb{F}_q^{r\times(n-r)} \times \mathbb{F}_{q^m}^{\rho\times 1}$ and $(\boldsymbol{s'}_{\mathsf{base}}, \boldsymbol{C'}_{\mathsf{base}}, \boldsymbol{v}_{\mathsf{base}}) \in \mathbb{F}_{q^m}^{r-1} \times \mathbb{F}_{q^m}^{r\times(n-r)} \times \mathbb{F}_{q^m}^{\rho\times 1}$.
4. One reveals the auxiliary value $\mathsf{aux} := (\boldsymbol{s'}_{\mathsf{aux}}, \boldsymbol{C'}_{\mathsf{aux}})$ with $\boldsymbol{s'}_{\mathsf{aux}} := \boldsymbol{s'} - \boldsymbol{s'}_{\mathsf{acc}}$ and $\boldsymbol{C'}_{\mathsf{aux}} = \boldsymbol{C'} - \boldsymbol{C'}_{\mathsf{acc}}$). Since $\boldsymbol{v}$ is a random vector (which is not part of the witness, it aims to mask the polynomial P_α), we do not need to rely on auxiliary value, we just define $\boldsymbol{v}$ as $\boldsymbol{v}_{\mathsf{acc}}$ (it is equivalent to say that $\boldsymbol{v}_{\mathsf{aux}} := 0$).
5. One defines $P_{s'}$, $P_{C'}$ and P_v respectively as $\boldsymbol{s'} \cdot X + \boldsymbol{s'}_{\mathsf{base}}$, $\boldsymbol{C'} \cdot X + \boldsymbol{C'}_{\mathsf{base}}$ and $\boldsymbol{v} \cdot X + \boldsymbol{v}_{\mathsf{base}}$.

Then, to open the evaluations $\boldsymbol{s'}_{\mathsf{Eval}} := P_{s'}(\phi(i^*))$, $\boldsymbol{C'}_{\mathsf{Eval}} := P_{C'}(\phi(i^*))$ and $\boldsymbol{v}_{\mathsf{Eval}} := P_v(\phi(i^*))$, the prover just reveals all the seeds $\{\mathsf{seed}_i\}_i$ except seed_{i^*}

and the verifier will be able to compute $\boldsymbol{s'}_{\mathsf{Eval}}$, $\boldsymbol{C'}_{\mathsf{Eval}}$ and $\boldsymbol{v}_{\mathsf{Eval}}$ as

$$\boldsymbol{s'}_{\mathsf{Eval}} = \phi(i^*) \cdot \boldsymbol{s'}_{\mathsf{aux}} + \sum_{i=1, i\neq i^*}^{N} (\phi(i^*) - \phi(i)) \cdot \boldsymbol{s'}_{\mathsf{rnd},i}$$

$$\boldsymbol{C'}_{\mathsf{Eval}} = \phi(i^*) \cdot \boldsymbol{C'}_{\mathsf{aux}} + \sum_{i=1, i\neq i^*}^{N} (\phi(i^*) - \phi(i)) \cdot \boldsymbol{C'}_{\mathsf{rnd},i}$$

$$\boldsymbol{v}_{\mathsf{Eval}} = \sum_{i=1, i\neq i^*}^{N} (\phi(i^*) - \phi(i)) \cdot \boldsymbol{v}_{\mathsf{rnd},i}$$

with $(\boldsymbol{s'}_{\mathsf{rnd},i}, \boldsymbol{C'}_{\mathsf{rnd},i}, \boldsymbol{v}_{\mathsf{rnd},i}) := \mathsf{PRG}(\mathsf{seed}_i)$.

One-Tree Optimization. Here above, we described how to commit the polynomials $P_{s'}$, $P_{C'}$ and P_v such that the prover can later open one evaluation. In practice, we perform this commitment phase τ times in parallel. It implies that we use τ all-but-one vector commitments, where each of them uses of GGM tree of N leaves. Therefore, we can optimize those all-but-one vector commitments by using a so-called batched all-but-one vector commitment (BAVC) scheme, which aims to be more efficient than τ independent AVC schemes. In RYDE, we use the BAVC scheme described in the article [BBM+25], which proposes the "one-tree" optimization. Instead of considering τ independent GGM trees of N leaves in parallel, the authors propose to rely on a unique large GGM tree of $\tau \cdot N$ leaves where the i^{th} seed of the e^{th} parallel repetition is associated to the $(e \cdot N + i)^{\text{th}}$ leaf of the large GGM tree. As explained in [BBM+25], "opening all but τ leaves of the big tree is more efficient than opening all but one leaf in each of the τ smaller trees, because with high probability some of the active paths in the tree will merge relatively close to the leaves, which reduces the number of internal nodes that need to be revealed." Moreover, the authors of [BBM+25] further propose to improve the previous approach using the principle of *grinding*. When the BAVC opening is such that the number of revealed nodes in the revealed sibling paths exceeds a chosen threshold T_{open}, the opening is considered as a failure (*i.e.* it returns $\perp$), forcing the prover/signer recomputing another opening challenge by hashing with an incremented counter. This process is done until the number of revealed nodes is less than T_{open}. For example, if we consider $N = 256$ and $\tau = 16$, the number of revealed nodes is smaller than (or equal to) $T_{\text{open}} := 110$ with probability ≈ 0.2. The selected value of T_{open} induces a rejection probability $p_{\text{rej}} = 1 - 1/\theta$, for some $\theta \in (0, \infty)$, and the signer hence needs to perform an average of θ hash computations for the opening challenge (instead of 1). While this strategy decreases the challenge space by a factor θ, it does not change the average number of hashes that must be computed to succeed an forgery attack against the signature scheme (since the latter is multiplied by θ). As noticed by the authors of [BBM+25], this strategy can be thought of as loosing $\log_2 \theta$ bit of security (because of a smaller challenge space) which are regained thanks to a proof-of-work (performing an average of θ hash computations before getting a valid challenge).

PIOP Computation. We now describe in more details the computation performed over the committed polynomials, namely the computation of P_α for the prover and the computation of $P_\alpha(r)$ for the verifier.

Prover's Computation. We detail here after the computation of the prover to build the polynomial P_x and the polynomial $P_\alpha(X)$. Let us denote $P_x := \boldsymbol{x} \cdot X^2 + \boldsymbol{x}_{\mathsf{mid}} \cdot X + \boldsymbol{x}_{\mathsf{base}}$ and $P_\alpha := \boldsymbol{\alpha} \cdot X^2 + \boldsymbol{\alpha}_{\mathsf{mid}} \cdot X + \boldsymbol{\alpha}_{\mathsf{base}}$. We also denote $\boldsymbol{s} = (1 \parallel \boldsymbol{s'})$ and $\boldsymbol{s}_{\mathsf{base}} = (1 \parallel \boldsymbol{s'}_{\mathsf{base}})$. The prover should compute $P_x(X) := (X \parallel P_{s'}(X)) \cdot [P_{I_r}(X) \parallel P_{C'}(X)]$, meaning that he should compute

$$\begin{aligned}
\boldsymbol{x} &:= [\boldsymbol{s} \parallel \boldsymbol{s} \cdot \boldsymbol{C'}] \\
\boldsymbol{x}_{\mathsf{mid}} &:= [\boldsymbol{s}_{\mathsf{base}} \parallel \boldsymbol{s}_{\mathsf{base}} \cdot \boldsymbol{C'} + \boldsymbol{s} \cdot \boldsymbol{C'}_{\mathsf{base}}] \\
\boldsymbol{x}_{\mathsf{base}} &:= [\boldsymbol{0}_{1\times r} \parallel \boldsymbol{s}_{\mathsf{base}} \cdot \boldsymbol{C'}_{\mathsf{base}}].
\end{aligned}$$

Then, the prover should compute the polynomial

$$P_\alpha(X) := P_v(X) + \boldsymbol{\Gamma} \cdot \left(P_x(X) \cdot (\boldsymbol{I}_{n-k} \parallel \boldsymbol{H'})^\top - \boldsymbol{y} \cdot X^2\right),$$

meaning that he should compute

$$\begin{aligned}
\boldsymbol{\alpha} &:= \boldsymbol{\Gamma} \cdot \left(\boldsymbol{x} \cdot (\boldsymbol{I}_{m\cdot n-k} \parallel \boldsymbol{H'})^\top - \boldsymbol{y}\right) \\
\boldsymbol{\alpha}_{\mathsf{mid}} &:= \boldsymbol{\Gamma} \cdot \boldsymbol{x}_{\mathsf{mid}} \cdot (\boldsymbol{I}_{m\cdot n-k} \parallel \boldsymbol{H'})^\top + \boldsymbol{v} \\
\boldsymbol{\alpha}_{\mathsf{base}} &:= \boldsymbol{\Gamma} \cdot \boldsymbol{x}_{\mathsf{base}} \cdot (\boldsymbol{I}_{m\cdot n-k} \parallel \boldsymbol{H'})^\top + \boldsymbol{v}_{\mathsf{base}}.
\end{aligned}$$

Let us remark that, by design, α is always zero so the prover does not need to compute it (by design, the polynomial P_α is of degree at most 1). Therefore, the prover does not need to compute $\boldsymbol{x}$: he just need to compute $\boldsymbol{x}_{\mathsf{mid}}$ and $\boldsymbol{x}_{\mathsf{base}}$ to build $P_\alpha(X) := \boldsymbol{\alpha}_{\mathsf{mid}} \cdot X + \boldsymbol{\alpha}_{\mathsf{base}}$.

Verifier's Computation. We detail here after the computation of the verifier to build the evaluations $\boldsymbol{x}_{\mathsf{Eval}} := P_x(r)$ and $\boldsymbol{\alpha}_{\mathsf{Eval}} := P_\alpha(r)$. The verifier should compute

$$\begin{aligned}
\boldsymbol{x}_{\mathsf{Eval}} &:= P_x(r) \\
&= [P_s(r)P_{I_r}(r) \parallel P_s(r)P_{C'}(r)] \\
&= [\boldsymbol{s}_{\mathsf{Eval}} \cdot r \parallel \boldsymbol{s}_{\mathsf{Eval}} \cdot \boldsymbol{C'}_{\mathsf{Eval}}],
\end{aligned}$$

together with

$$\begin{aligned}
\boldsymbol{\alpha}_{\mathsf{Eval}} &:= P_\alpha(r) \\
&= P_v(r) + \boldsymbol{\Gamma} \cdot \left((\boldsymbol{I}_{m\cdot n-k} \parallel \boldsymbol{H'}) \cdot P_x(r) - \boldsymbol{y} \cdot r^2\right) \\
&= \boldsymbol{v}_{\mathsf{Eval}} + \boldsymbol{\Gamma} \cdot \left((\boldsymbol{I}_{m\cdot n-k} \parallel \boldsymbol{H'}) \cdot \boldsymbol{x}_{\mathsf{Eval}} - \boldsymbol{y} \cdot r^2\right)
\end{aligned}$$

where $P_s = (X \parallel P_{s'})$ and $\boldsymbol{s}_{\mathsf{Eval}} = (r \parallel \boldsymbol{s'}_{\mathsf{Eval}})$.

Fiat-Shamir Transformation & Grinding. To obtain the RYDE signature scheme from the RYDE protocol, we rely on the Fiat-Shamir transformation [FS87] to remove the prover-verifier interactions. Each verifier's challenge is computed as the output of an extendable-output function (XOF) which takes as input the data that the prover would send before receiving that challenge in an interactive protocol. The RYDE protocol is a 5-round proof system, so there are two challenges: the randomness $\boldsymbol{\Gamma}$ involved in the definition of P_α, and the evaluation points onto which all the polynomials are evaluated. Since the signature scheme is the non-interactive variant of a 5-round protocol repeated τ times in parallel, the scheme is affected by the attack of Kales and Zaverucha [KZ20]. To avoid incrementing the number τ of parallel protocol repetitions (to have a secure scheme), we draw the first challenge from an exponentially-large set. Therefore, this challenge might be the same across all the parallel repetitions. Using this strategy, to have a secure scheme, $q^{-m\cdot\rho}$ and $(2/N)^\tau$ should be negligible. In the RYDE scheme, we also use a (explicit) proof-of-work to the Fiat-Shamir hash computation of the last challenge, as proposed in [BBM+25]. Together with the opening challenge, the signer samples a w-bit value v_{grinding} and keeps the opening challenge only if this additional value is zero, with w a parameter of the scheme. If this additional value is not zero, then the signer increments a counter and recompute an other opening challenge with an other w-bit value, and he repeats the process until the grinding value is zero. Let us remark that we can use the same counter for this grinding process and the grinding process due to the fact that the [BBM+25]'s BAVC scheme might return $\perp$ when the number of revealed nodes is larger than the chosen threshold T_{open}. This strategy increases the cost of hashing the last challenge by a factor 2^w and hence increases the security of w bits. This thus allows to take smaller parameters (N, τ) for the large tree, namely parameters achieving $\lambda - w$ bits of security instead of λ. More precisely, the parameters N, τ and w will be chosen such that $(2/N)^\tau \cdot 2^{-w} \leq 2^{-\lambda}$ to achieve a λ-bit security.

We describe the resulting signature scheme in Algorithms 1 and 2. We added a random salt to have a domain separation between signatures. Let us remark that RYDE uses standard techniques to optimize the signature size: instead of including all the prover's sent data, the signature only contains minimal information that enables the verifier to recompute the prover's sent data and will check if the Fiat-Shamir hashes are consistent with the recomputed data. More precisely,

- Instead of including the evaluations $\boldsymbol{s}'_{\mathsf{Eval}}$, $\boldsymbol{C}'_{\mathsf{Eval}}$ and $\boldsymbol{v}_{\mathsf{Eval}}$, the signature includes information (data enabling to derive $\{\mathsf{seed}_i\}_{i\neq i^*}$ and the auxiliary value $(\boldsymbol{s}'_{\mathsf{aux}}, \boldsymbol{C}'_{\mathsf{aux}})$) that enables the verifier to derive them.
- Instead of including the polynomial $P_\alpha := \boldsymbol{\alpha}_{\mathsf{mid}} \cdot X + \boldsymbol{\alpha}_{\mathsf{base}}$, the signature only contains $\boldsymbol{\alpha}_{\mathsf{mid}}$. Then, using $\boldsymbol{\alpha}_{\mathsf{Eval}} := P_\alpha(\phi(i^*))$ and $\boldsymbol{\alpha}_{\mathsf{mid}}$, the verifier will be able to recompute $\boldsymbol{\alpha}_{\mathsf{base}}$.

Public key: A Rank Syndrome Decoding instance $(\boldsymbol{H}, \boldsymbol{y})$.

Secret key: A vector $\boldsymbol{s'} \in \mathbb{F}_{q^m}^{r-1}$ and $\boldsymbol{C'} \in \mathbb{F}_q^{r \times r(n-r)}$ satisfying $(1 \parallel \boldsymbol{s'}) \cdot [I_r \parallel \boldsymbol{C'}]) = \boldsymbol{y}$.

Step 0: Initialization

1. Uncompress the public and secret keys (if they are in a compressed form).
2. Sample a random $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\lambda}$.

Step 1: Build & commit to witness polynomials

3. Using a (salted) BAVC, derive τ sets of N seeds $\{\mathsf{seed}_1^{(e)}, \ldots, \mathsf{seed}_N^{(e)}\}_e$ with their commitment digests $\mathsf{com}_{e,i}$.
4. For each iteration $e \in [1, \ldots, \tau]$:
   (a) For all $i \in [1, \ldots, N]$, expand each seed $\mathsf{seed}_i^{(e)}$ as $(\boldsymbol{s'}_{\mathsf{rnd},i}^{(e)}, \boldsymbol{C'}_{\mathsf{rnd},i}^{(e)}, \boldsymbol{v}_{\mathsf{rnd},i}^{(e)})$.
   (b) Compute $\boldsymbol{s'}_{\mathsf{acc}}^{(e)} = \sum_{i=1}^{N} \boldsymbol{s'}_{\mathsf{rnd},i}^{(e)}$, $\boldsymbol{C'}_{\mathsf{acc}}^{(e)} = \sum_{i=1}^{N} \boldsymbol{C'}_{\mathsf{rnd},i}^{(e)}$, and $\boldsymbol{v}_{\mathsf{acc}}^{(e)} = \sum_{i=1}^{N} \boldsymbol{v}_{\mathsf{rnd},i}^{(e)}$.
   (c) Compute $\boldsymbol{s'}_{\mathsf{base}}^{(e)} = -\sum_{i=1}^{N} \phi(i) \cdot \boldsymbol{s'}_{\mathsf{rnd},i}^{(e)}$, and $\boldsymbol{C'}_{\mathsf{base}}^{(e)} = -\sum_{i=1}^{N} \phi(i) \cdot \boldsymbol{C'}_{\mathsf{rnd},i}^{(e)}$, and $\boldsymbol{v}_{\mathsf{base}}^{(e)} = -\sum_{i=1}^{N} \phi(i) \cdot \boldsymbol{v}_{\mathsf{rnd},i}^{(e)}$.
   (d) Compute $\boldsymbol{s'}_{\mathsf{aux}}^{(e)} = \boldsymbol{s'} - \boldsymbol{s'}_{\mathsf{acc}}^{(e)}$ and $\boldsymbol{C'}_{\mathsf{aux}}^{(e)} = \boldsymbol{C'} - \boldsymbol{C'}_{\mathsf{acc}}^{(e)}$, and set $\boldsymbol{v}^{(e)}$ as $\boldsymbol{v}_{\mathsf{acc}}^{(e)}$.
5. Compute $h_1 = \mathsf{Hash}_1(\mathsf{salt}, \{\mathsf{com}_{e,i}\}_{e \in [1,\ldots,\tau], i \in [1,\ldots,N]}, (\boldsymbol{s'}_{\mathsf{aux}}^{(e)}, \boldsymbol{C'}_{\mathsf{aux}}^{(e)})_{e \in [1,\ldots,\tau]})$

Step 2: Compute the polynomial proof $P_\alpha(X)$

6. Sample $\boldsymbol{\Gamma} \xleftarrow{\$} \mathsf{PRG}(h_{\mathrm{sh}})$ where $\boldsymbol{\Gamma} \in \mathbb{F}_{q^m}^{(n-k) \times \rho}$.
7. For each iteration $e \in [1, \ldots, \tau]$:
   (a) Compute $P_x^{(e)}(X)$ by computing

$$\boldsymbol{x}_{\mathsf{mid}}^{(e)} := [\boldsymbol{s}_{\mathsf{base}}^{(e)} \parallel \boldsymbol{s}_{\mathsf{base}}^{(e)} \cdot \boldsymbol{C'}^{(e)} + \boldsymbol{s}^{(e)} \cdot \boldsymbol{C'}_{\mathsf{base}}^{(e)}],$$
$$\boldsymbol{x}_{\mathsf{base}}^{(e)} := [\boldsymbol{0}_{m \times r} \parallel \boldsymbol{s}_{\mathsf{base}}^{(e)} \cdot \boldsymbol{C'}_{\mathsf{base}}^{(e)}].$$

   where $\boldsymbol{s}_{\mathsf{base}}^{(e)} = (1 \parallel \boldsymbol{s'}_{\mathsf{base}}^{(e)})$.
   (b) Compute the polynomial $P_\alpha^{(e)} := \boldsymbol{\alpha}_{\mathsf{mid}}^{(e)} \cdot X + \boldsymbol{\alpha}_{\mathsf{base}}^{(e)}$ by computing

$$\boldsymbol{\alpha}_{\mathsf{mid}}^{(e)} := \boldsymbol{x}_{\mathsf{mid}}^{(e)} \cdot (\boldsymbol{I}_{m \cdot n - k} \parallel \boldsymbol{H'})^\top \cdot \boldsymbol{\Gamma} + \boldsymbol{v}^{(e)},$$
$$\boldsymbol{\alpha}_{\mathsf{base}}^{(e)} := \boldsymbol{x}_{\mathsf{base}}^{(e)} \cdot (\boldsymbol{I}_{m \cdot n - k} \parallel \boldsymbol{H'})^\top \cdot \boldsymbol{\Gamma} + \boldsymbol{v}_{\mathsf{base}}^{(e)}.$$

8. Compute $h_2 = \mathsf{Hash}_2\left(\mathsf{Hash}_0(\mathsf{msg}), \mathsf{pk}, \mathsf{salt}, h_1, (\boldsymbol{\alpha}_{\mathsf{mid}}^{(e)}, \boldsymbol{\alpha}_{\mathsf{base}}^{(e)})_{e \in [1,\ldots,\tau]}\right)$.

Step 3: Open random evaluations

9. Set $\mathsf{ctr} := 0$.
10. Sample $\left(v_{\mathrm{grinding}}, \{i^{*(e)}\}_e\right) \xleftarrow{\$} \mathsf{PRG}(h_2, \mathsf{ctr})$ where $i^{*(e)} \in [1, \ldots, N]$ for all $e \in [1, \ldots, \tau]$ and $v_{\mathrm{grinding}} \in \{0,1\}^w$.
11. Compute the BAVC's opening proof π_{BAVC} for $\{\mathsf{seed}_i^e\}_{e, i \neq i^{*(e)}}$.
12. If $v_{\mathrm{grinding}} \neq 0$ or $\pi_{\mathsf{BAVC}} = \perp$, increment ctr and go to Step 10.
13. Output the signature $\sigma = \left(\mathsf{salt} \mid \mathsf{ctr} \mid h_2 \mid \pi_{\mathsf{BAVC}} \mid \left(\boldsymbol{s'}_{\mathsf{aux}}^{(e)}, \boldsymbol{C'}_{\mathsf{aux}}^{(e)}, \boldsymbol{\alpha}_{\mathsf{mid}}^{(e)}\right)_{e \in [1,\ldots,\tau]}\right)$.

Fig. 1. High level description of RYDE Sign algorithm

Public key: A Rank Syndrome Decoding instance $(\boldsymbol{H}, \boldsymbol{y})$.

Step 0: Initialization

1. Uncompress the public key (if it is in a compressed form).
2. Parse the signature as $\left(\mathsf{salt} \mid \mathsf{ctr} \mid h_2 \mid \pi_{\mathsf{BAVC}} \mid \left(\boldsymbol{s'}^{(e)}_{\mathsf{aux}}, \boldsymbol{C'}^{(e)}_{\mathsf{aux}}, \boldsymbol{\alpha}^{(e)}_{\mathsf{mid}}\right)_{e\in[1,\dots,\tau]}\right)$.

Step 1: Computing opened evaluations

3. Sample $\left(v_{\text{grinding}}, \{i^{*(e)}\}_e\right) \xleftarrow{\$} \mathsf{PRG}(h_{\text{piop}}, \mathsf{ctr})$ where $i^{*(e)} \in [1,\dots,N]$ for all $e \in [1,\dots,\tau]$ and $v \in \{0,1\}^w$.
4. Using π_{BAVC} and $\{i^{*(e)}\}_e$, recover $\{\mathsf{seed}^e_i\}_{e,i\neq i^{*(e)}}$ with the reconstruction algorithm of the BAVC scheme, together with the commitment digests $\{\mathsf{com}_{e,i}\}_{e\in[1,\dots,\tau],i\in[1,\dots,N]}$.
5. For each iteration $e \in [1,\dots,\tau]$:
   (a) For all $i \in [1,\dots,N]\backslash\{i^{*(e)}\}$, expand each seed $\mathsf{seed}^{(e)}_i$ as $(\boldsymbol{s'}^{(e)}_{\mathsf{rnd},i}, \boldsymbol{C'}^{(e)}_{\mathsf{rnd},i}, \boldsymbol{v}^{(e)}_{\mathsf{rnd},i})$.
   (b) Compute

$$\boldsymbol{s'}^{(e)}_{\mathsf{Eval}} = \phi(i^{*(e)}) \cdot \boldsymbol{s'}^{(e)}_{\mathsf{aux}} + \sum_{i=1,i\neq i^{*(e)}}^{N} (\phi(i^{*(e)}) - \phi(i)) \cdot \boldsymbol{s'}^{(e)}_{\mathsf{rnd},i}$$

$$\boldsymbol{C'}^{(e)}_{\mathsf{Eval}} = \phi(i^{*(e)}) \cdot \boldsymbol{C'}^{(e)}_{\mathsf{aux}} + \sum_{i=1,i\neq i^{*(e)}}^{N} (\phi(i^{*(e)}) - \phi(i)) \cdot \boldsymbol{C'}^{(e)}_{\mathsf{rnd},i}$$

$$\boldsymbol{v}^{(e)}_{\mathsf{Eval}} = \sum_{i=1,i\neq i^{*(e)}}^{N} (\phi(i^{*(e)}) - \phi(i)) \cdot \boldsymbol{v}^{(e)}_{\mathsf{rnd},i} \ .$$

6. Compute $h_1 = \mathsf{Hash}_1(\mathsf{salt}, \{\mathsf{com}_{e,i}\}_{e\in[1,\dots,\tau],i\in[1,\dots,N]}, (\boldsymbol{s'}^{(e)}_{\mathsf{aux}}, \boldsymbol{C'}^{(e)}_{\mathsf{aux}})_{e\in[1,\dots,\tau]})$

Step 2: Recompute the polynomial proof $P_\alpha(X)$

7. Sample $\boldsymbol{\Gamma} \xleftarrow{\$} \mathsf{PRG}(h_{\text{sh}})$ where $\boldsymbol{\Gamma} \in \mathbb{F}_{q^m}^{(n-k)\times\rho}$.
8. For each iteration $e \in [1,\dots,\tau]$:
   (a) Compute the evaluation $\boldsymbol{x}^{(e)}_{\mathsf{Eval}} := P^{(e)}_x(\phi(i^{*(e)}))$ as

$$\boldsymbol{x}^{(e)}_{\mathsf{Eval}} = [\boldsymbol{s}^{(e)}_{\mathsf{Eval}} \cdot \phi(i^{*(e)}) \parallel \boldsymbol{s}^{(e)}_{\mathsf{Eval}} \cdot \boldsymbol{C'}^{(e)}_{\mathsf{Eval}}].$$

   where $\boldsymbol{s}^{(e)}_{\mathsf{Eval}} = \left(\phi(i^{*(e)}) \parallel \boldsymbol{s'}^{(e)}_{\mathsf{Eval}}\right)$.
   (b) Compute the evaluation $\boldsymbol{\alpha}^{(e)}_{\mathsf{Eval}} := P^{(e)}_\alpha(\phi(i^{*(e)}))$ as

$$\boldsymbol{\alpha}^{(e)}_{\mathsf{Eval}} = \boldsymbol{v}_{\mathsf{Eval}} + \left(\boldsymbol{x}^{(e)}_{\mathsf{Eval}} \cdot (\boldsymbol{I}_{m\cdot n-k} \parallel \boldsymbol{H'})^\top - \boldsymbol{y} \cdot \phi(i^{*(e)})^2\right) \cdot \boldsymbol{\Gamma}.$$

   (c) Deduce $\boldsymbol{\alpha}^{(e)}_{\mathsf{base}}$ as

$$\boldsymbol{\alpha}^{(e)}_{\mathsf{base}} = \boldsymbol{\alpha}^{(e)}_{\mathsf{Eval}} - \boldsymbol{\alpha}^{(e)}_{\mathsf{mid}} \cdot \phi(i^{*(e)}).$$

9. Compute $h'_2 = \mathsf{Hash}_2\left(\mathsf{Hash}_0(\mathsf{msg}), \mathsf{pk}, \mathsf{salt}, h_1, (\boldsymbol{\alpha}^{(e)}_{\mathsf{mid}}, \boldsymbol{\alpha}^{(e)}_{\mathsf{base}})_{e\in[1,\dots,\tau]}\right)$.

Step 3: Verification

10. Check that $h'_2 =_? h_2$ and $v_{\text{grinding}} = 0$.

Fig. 2. High level description of RYDE Verify algorithm

4 Algorithmic Description

4.1 Notations

We let $y \leftarrow \mathsf{A}(x)$ denote the operation that runs algorithm A on input x and assigns the output to the variable y. The notation $a \mathrel{+}= b$ means that to the variable a it is assigned the value $a + b$, where the addition is performed component-wise if a and b are tuples. We employ arrays, and we let $\mathsf{a}[i]$ denote the ith element of the array a. The main variable names employed in the algorithms are collected in Table 2.

Seeds:		
$\mathsf{seed}_{\mathsf{pk}}$	$\{0,1\}^{\lambda}$	Seed for the generation of the matrix $\boldsymbol{H}$ of the public key.
$\mathsf{seed}_{\mathsf{sk}}$	$\{0,1\}^{\lambda}$	Seed for the generation of the matrices $\boldsymbol{s'}, \boldsymbol{C}$ of the secret key.
rseed	$\{0,1\}^{\lambda}$	Root seed for the seed tree.
seeds	$(\{0,1\}^{\lambda})^{N}$	Leaf seeds for the seed tree.
salt	$\{0,1\}^{2\lambda}$	Salt.
salt_i	$\{0,1\}^{\lambda}$	The i-th half of the salt.
tree	$(\{0,1\}^{\lambda})^{2N}$	Nodes of the seed tree.
path	$(\{0,1\}^{\lambda})^{*}$	Part of the seed tree revealed to the verifier.
Vectors and Matrices:		
$\boldsymbol{H}$	$\mathbb{F}_{q^m}^{(n-k)\times k}$	Matrix of the public key.
$\boldsymbol{x}$	$\mathbb{F}_{q^m}^{n}$	Secret error.
$\boldsymbol{y}$	$\mathbb{F}_{q^m}^{n-k}$	Public syndrome vector.
$\boldsymbol{s'}$	$\mathbb{F}_{q^m}^{r-1}$	Secret support.
$\boldsymbol{C}$	$\mathbb{F}_{q}^{r\times(n-r)}$	Secret coordinates matrix.
$\boldsymbol{v}$	$\mathbb{F}_{q^m}^{\rho}$	Masking vector.
$\boldsymbol{\Gamma}$	$\mathbb{F}_{q^m}^{(n-k)\times\rho}$	Challenge matrix.
$\boldsymbol{\alpha}$	$\mathbb{F}_{q^m}^{\rho}$	Response vector to challenge $\boldsymbol{\Gamma}$
Polynomial coefficients and evaluations:		
point	$\mathbb{F}_{q^m}$	Evaluation scalar.
base_u		Degree-0 coefficient of the polynomial representing the vector $\boldsymbol{u}$ (in its leading term).
mid_u		Degree-1 coefficient of the polynomial representing $\boldsymbol{u}$.
share_u		Evaluation at point of the polynomial representing $\boldsymbol{u}$.
acc_u		Intermediary variable to compute aux_u
aux_u		Correcting term between the sum of expanded seeds for $\boldsymbol{u}$ and its actual secret value.
Others:		
msg	$\{0,1\}^{*}$	The message the be signed.

Table 2. Algorithmic notation.

4.2 Finite Fields Representation

Every element in $\mathbb{F}_{q^m}$ is represented by a degree-$(m-1)$ polynomial $a_{m-1}x^{m-1} + \cdots + a_1x + a_0$, where $a_0, \ldots, a_{m-1} \in \mathbb{F}_q$. We store an element $a_{m-1}x^{m-1} + \cdots + a_1x + a_0 \in \mathbb{F}_{q^m}$ as the integer with binary representation

$$\texttt{bin}(a_{m-1}) \parallel \texttt{bin}(a_{m-2}) \parallel \cdots \parallel \texttt{bin}(a_1) \parallel \texttt{bin}(a_0),$$

where $\texttt{bin}(a_i)$ is the binary representation of $a_i \in \mathbb{F}_q$. We use $(m \log q)$ bits to store a single $\mathbb{F}_{q^m}$-element. The multiplication of two elements in $\mathbb{F}_{q^m}$ is implemented as a polynomial multiplication modulo an irreducible polynomial $f(x)$, where $f(x)$ is chosen as given in Table 3.

q	m	$f(x)$
2	53	$x^{53} + x^6 + x^2 + x + 1$
2	61	$x^{61} + x^5 + x^2 + x + 1$
2	67	$x^{67} + x^5 + x^2 + x + 1$

Table 3. Polynomial modulus $f(x)$ for multiplications in $\mathbb{F}_{q^m}$.

4.3 Randomness Generation and Sampling

In the following, we describe the routines used for generating pseudorandom objects.

ExpandSecret(seed) $\triangleright$ $\mathbb{F}_{q^m}^{r-1} \times \mathbb{F}_q^{r\times(n-r)}$*:* From a seed, generates a secret key pair $(\boldsymbol{s'}, \boldsymbol{C})$, where $\boldsymbol{s'}$ is a vector such that the support $\boldsymbol{s} = (1 \parallel \boldsymbol{s'}) \in \mathbb{F}_{q^m}^r$ verifies $\mathsf{rank}(\boldsymbol{s}) = r$; and $\boldsymbol{C}$ a coordinate matrix sampled uniformly. For the security level parameter $\lambda = 128$, we use $\mathsf{SHAKE128}$ with input seed. While for $\lambda \in \{192, 256\}$, we use $\mathsf{SHAKE256}$.

ExpandMatrixH(seed) $\triangleright$ $\mathbb{F}_{q^m}^{(n-k)\times k}$*:* From a seed, samples a public matrix $\boldsymbol{H}$ uniformly. For the security level parameter $\lambda = 128$, we use $\mathsf{SHAKE128}$ with input seed. While for $\lambda \in \{192, 256\}$, we use $\mathsf{SHAKE256}$.

ExpandSeed($\mathsf{seed}, \mathsf{salt}, i$) $\triangleright$ $\{0,1\}^\lambda \times \{0,1\}^\lambda$*:* This routine is used in the GGM tree expansion. It generates the two children of a parent $\mathsf{node}[i] = \mathsf{seed}$ using $\mathsf{salt} = (\mathsf{salt}_0 \,||\, \mathsf{salt}_1)$ and i as additional randomness. More precisely,

$$\mathsf{node}[2i] = \begin{cases} \mathsf{AES\text{-}128}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus (\mathsf{dom} \,||\, i \,||\, \mathsf{0x00})) & \text{if } \lambda = 128 \\ \mathsf{Rijndael\text{-}256}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus (\mathsf{dom} \,||\, i \,||\, \mathsf{0x00})) & \text{otherwise.} \end{cases}$$

$$\mathsf{node}[2i+1] = \begin{cases} \mathsf{AES\text{-}128}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus (\mathsf{dom} \,||\, i \,||\, \mathsf{0x01})) & \text{if } \lambda = 128 \\ \mathsf{Rijndael\text{-}256}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus (\mathsf{dom} \,||\, i \,||\, \mathsf{0x01})) & \text{otherwise.} \end{cases}$$

For the security level parameter $\lambda = 192$, we truncate the output of Rijndael-256 to 192 bits. In all the three cases, we take $\mathsf{dom} = \mathsf{0x04}$, and we represent $i \in \{0,1\}^{32}$ as byte string of length eight.

$\mathsf{ExpandShare}(\mathsf{seed}, \mathsf{salt}) \triangleright \mathbb{F}_{q^m}^{r-1} \times \mathbb{F}_q^{r\times(n-r)} \times \mathbb{F}_{q^m}^{\rho}$*:* From a seed, samples a triple $(\boldsymbol{s'}, \boldsymbol{C}, \boldsymbol{v})$ uniformly, using a $\mathsf{salt} = (\mathsf{salt}_0 \mid\mid \mathsf{salt}_1)$ as extra randomness. We use AES-128 (resp. Rijndael-256) in CTR mode. For the security level parameter $\lambda = 192$, we truncate the output of Rijndael-256 to 192 bits. In all the three cases, the counter ctr goes from 0 to $\mathsf{block\_length} - 1$, where

$$\mathsf{block\_length} = \left\lceil \frac{\mathsf{bytes}(\boldsymbol{s'}) + \mathsf{bytes}(\boldsymbol{C}) + \mathsf{bytes}(\boldsymbol{v})}{\lambda} \right\rceil .$$

We sample $(\mathsf{block\_length} \cdot \lambda)$ bits as $\mathsf{blocks} = (\mathsf{block}[0] \mid\mid \cdots \mid\mid \mathsf{block}[\mathsf{block\_length} - 1])$ with

$$\mathsf{block}[\mathsf{ctr}] = \begin{cases} \mathsf{AES\text{-}128}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus \mathsf{ctr}) & \text{if } \lambda = 128 \\ \mathsf{Rijndael\text{-}256}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus \mathsf{ctr}) & \text{otherwise.} \end{cases}$$

We truncate the bits of blocks to get the exact amount of bytes representing $\boldsymbol{s'}$, $\boldsymbol{C}$, and $\boldsymbol{v}$.

$\mathsf{ExpandChallenge1}(\mathsf{seed}) \triangleright \mathbb{F}_{q^m}^{(n-k)\times\rho}$*:* From a seed, samples a challenge $\boldsymbol{\Gamma}$ uniformly. For the security level parameter $\lambda = 128$, we use SHAKE128 with input seed. While for $\lambda \in \{192, 256\}$, we use SHAKE256.

$\mathsf{ExpandChallenge2}(\mathsf{seed}) \triangleright [1, N_1]^{\tau_1} \times [1, N_2]^{\tau_2} \times \{0,1\}^w$*:* From a seed, samples an array of revealed indexes $\boldsymbol{i^*}$ and the grinding variable $v_{grinding}$ uniformly. For the security level parameter $\lambda = 128$, we use SHAKE128 with input seed. While for $\lambda \in \{192, 256\}$, we use SHAKE256.

4.4 Hash Functions and Commitments

We instantiate our hash functions with SHA3-λ as follows:

$$\begin{aligned} \mathsf{Hash}_0(\mathsf{data}) &\coloneqq \mathsf{SHA3\text{-}}\lambda(\mathsf{0x00}, \mathsf{data}), \\ \mathsf{Hash}_1(\mathsf{data}) &\coloneqq \mathsf{SHA3\text{-}}\lambda(\mathsf{0x01}, \mathsf{data}), \text{ and} \\ \mathsf{Hash}_2(\mathsf{data}) &\coloneqq \mathsf{SHA3\text{-}}\lambda(\mathsf{0x02}, \mathsf{data}). \end{aligned}$$

For the commitments, we have the following two possible flavors.

Rijndael-*based commitment:* Let $j = (N + i)$ and $\mathsf{salt} = (\mathsf{salt}_0 \mid\mid \mathsf{salt}_1)$, then we calculate $\mathsf{Commit}(\mathsf{salt}, i, \mathsf{seed}) \coloneqq (\mathsf{high} \mid\mid \mathsf{low})$ where

$$\mathsf{high} = \begin{cases} \mathsf{AES\text{-}128}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus (\mathsf{dom} \mid\mid j \mid\mid \mathsf{0x00})) & \text{if } \lambda = 128 \\ \mathsf{Rijndael\text{-}256}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus (\mathsf{dom} \mid\mid j \mid\mid \mathsf{0x00})) & \text{otherwise.} \end{cases}$$

$$\mathsf{low} = \begin{cases} \mathsf{AES\text{-}128}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus (\mathsf{dom} \mid\mid j \mid\mid \mathsf{0x01})) & \text{if } \lambda = 128 \\ \mathsf{Rijndael\text{-}256}(\mathsf{k} = \mathsf{seed}, \mathsf{msg} = \mathsf{salt}_0 \oplus (\mathsf{dom} \mid\mid j \mid\mid \mathsf{0x01})) & \text{otherwise.} \end{cases}$$

For the security level parameter $\lambda = 192$, we truncate the output of Rijndael-256 to 192 bits. In all the three cases, we take $\mathsf{dom} = \mathsf{0x03}$, and we represent $j \in \{0,1\}^{32}$ as byte string of length eight.

SHA3-*based commitment:*

$$\mathsf{Commit}(\mathsf{salt}, i, \mathsf{seed}) := \mathsf{SHA3}\text{-}\lambda(\mathsf{0x03}, \mathsf{salt}, i, \mathsf{seed}).$$

4.5 Tree Routines

The seeds have a tree structure, where $\mathsf{seeds} = \{\mathsf{seeds}[e][i]\}_{e<\tau, i<N_e}$ is a set of $N = \sum_{e=1}^{\tau} N_e$ elements in $\{0,1\}^{\lambda}$. The N_e's are positive integers such that $N_1 = \cdots = N_{\tau_1} \geq N_{\tau_1+1} = \cdots = N_\tau$ for some integer $\tau_1 > 0$. In addition, it uses the function ψ defined as

$$\psi(e,i) := \begin{cases} (i-1)\cdot\tau + (e-1) + 1 & \text{if } i \leq N_\tau \\ N_\tau \cdot \tau + (i - N_\tau - 1)\cdot\tau_1 + (e-1) + 1 & \text{otherwise.} \end{cases} \tag{4}$$

Algorithm 1 Routine Tree.PRG.

```
Tree.PRG(salt, rseed)
1 : tree[0] ← rseed
2 : for i from 0 to (N − 1) do
3 :     (tree[2i + 1], tree[2i + 2]) ← ExpandSeed(salt, tree[i], i)
4 : return tree
```

Algorithm 2 Routine Tree.GetSiblingPath.

```
Tree.GetSiblingPath(tree, {i*[e]}_e)
1 : hidden ← {N + ψ(e, i*[e]) : e ∈ {0, . . . , τ − 1}}
2 : revealed ← {N, . . . , 2N − 1}\hidden
3 : for i from (N − 1) downto 1 do
4 :     if (2i) ∈ revealed and (2i + 1) ∈ revealed then
5 :         revealed ← (revealed\{2i, 2i + 1}) ∪ {i}
6 : path ← ∅
7 : for i from 1 to 2N − 1 do
8 :     if i ∈ revealed then
9 :         path ← (path ‖ tree[i])
10 : return path
```

Algorithm 3 Routine Tree.GetSeedsFromPath.

Tree.GetSeedsFromPath$(\{i^*[e]\}_e, \mathsf{pdecom}, \mathsf{salt})$

1 : $\mathsf{hidden} \leftarrow \{N + \psi(e, i^*[e]) : e \in \{0, \ldots, \tau - 1\}\}$
2 : $\mathsf{revealed} \leftarrow \{N, \ldots, 2N-1\} \backslash \mathsf{hidden}$
3 : **for** i from $(N-1)$ downto 1 **do**
4 : **if** $(2i) \in \mathsf{revealed}$ and $(2i+1) \in \mathsf{revealed}$ **then**
5 : $\mathsf{revealed} \leftarrow (\mathsf{revealed} \backslash \{2i, 2i+1\}) \cup \{i\}$
6 : $\mathsf{tree}[1], \ldots, \mathsf{tree}[2N-1] \leftarrow \emptyset, \ldots, \emptyset$
7 : **for** i from 1 to $(N-1)$ **do**
8 : **if** $i \in \mathsf{revealed}$ **then**
9 : $(\mathsf{tree}[i], \mathsf{path}) \leftarrow \mathsf{path}$
10 : **if** $\mathsf{tree}[i] \neq \emptyset$ **then**
11 : $(\mathsf{tree}[2i+1], \mathsf{tree}[2i+2]) \leftarrow \mathrm{ExpandSeed}(\mathsf{salt}, \mathsf{tree}[i], i)$
12 : **for** e from 0 to $(\tau - 1)$ **do**
13 : **for** i from 0 to $(N_e - 1)$ **do**
14 : **if** $i \neq i^*[e]$ **then**
15 : $\mathsf{seeds}[e][i] \leftarrow \mathsf{tree}[N + \psi(e, i)]$ ▷ $\psi(\cdot,\cdot)$ defined in Equation (4)
16 : **else**
17 : $\mathsf{seeds}[e][i] \leftarrow \emptyset$
18 : **return** $\{\mathsf{seeds}[e][i]\}_{e,i}$

4.6 Polynomial Computation Routines

In the polynomial computations, we split the matrix $\boldsymbol{C} \in \mathbb{F}_q^{r \times (n-r)}$ into $\boldsymbol{C}_1 \in \mathbb{F}_q^{1 \times (n-r)}$ and $\boldsymbol{C'} \in \mathbb{F}_q^{(r-1) \times (n-r)}$ such that $\boldsymbol{C} = \begin{pmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C'} \end{pmatrix}$. The same split is applied to the corresponding polynomial evaluations base_C and share_C.

Algorithm 4 Routine ComputePolynomialProof.

ComputePolynomialProof$(\mathsf{base}, \boldsymbol{v}, \boldsymbol{s'}, \boldsymbol{C}, \boldsymbol{\Gamma}, \boldsymbol{H})$

1 : $(\mathsf{base}_{s'}, \mathsf{base}_C, \mathsf{base}_v) \leftarrow \mathsf{base}$
2 : $(\mathsf{base}_x^L \parallel \mathsf{base}_x^R) \leftarrow \mathsf{base}_{s'} \cdot \mathsf{base}_{C'}$ ▷ $(\mathsf{base}_x^L \parallel \mathsf{base}_x^R) \in \mathbb{F}_{q^m}^{n-r-k} \times \mathbb{F}_{q^m}^k$
3 : $\mathsf{base}_\alpha \leftarrow [(0_r \parallel \mathsf{base}_x^L) + \mathsf{base}_x^R \cdot H^T] \cdot \Gamma + \mathsf{base}_v$ ▷ $\mathsf{base}_\alpha \in \mathbb{F}_{q^m}^\rho$
4 : $(\mathsf{mid}_x^L \parallel \mathsf{mid}_x^R) \leftarrow \left(\mathsf{base}_{s'} \parallel \mathsf{base}_{C_1} + s' \mathsf{base}_{C'} + \mathsf{base}_{s'} C'\right)$
5 : ▷ $(\mathsf{mid}_x^L \parallel \mathsf{mid}_x^R) \in \mathbb{F}_{q^m}^{n-1-k} \times \mathbb{F}_{q^m}^k$
6 : $\mathsf{mid}_\alpha \leftarrow [(0 \parallel \mathsf{mid}_x^L) + \mathsf{mid}_x^R \cdot H^T] \cdot \Gamma + v$ ▷ $\mathsf{mid}_\alpha \in \mathbb{F}_{q^m}^\rho$
7 : **return** $(\mathsf{mid}_\alpha, \mathsf{base}_\alpha)$

Algorithm 5 Routine RecomputePolynomialProof.

RecomputePolynomialProof$(\mathsf{point}, \mathsf{share}, \boldsymbol{\Gamma}, \boldsymbol{H}, \boldsymbol{y}, \mathsf{mid}_\alpha)$

1 : $(\mathsf{share}_{s'}, \mathsf{share}_C, \mathsf{share}_v) \leftarrow \mathsf{share}$
2 : $\mathsf{tmp}_x \leftarrow \mathsf{share}_{C_1} \cdot \mathsf{point} + \mathsf{share}_{s'} \cdot \mathsf{share}_{C'}$
3 : $(\mathsf{share}_x^L \parallel \mathsf{share}_x^R) \leftarrow (\mathsf{share}_{s'} \cdot \mathsf{point} \parallel \mathsf{tmp}_x)$ $\quad \triangleright\ \mathsf{share}_x \in \mathbb{F}_{q^m}^{n-1-k} \times \mathbb{F}_{q^m}^{k}$
4 : $\mathsf{share}_\alpha \leftarrow [(\mathsf{point}^2 \parallel \mathsf{share}_x^L) + \mathsf{share}_x^R \cdot H^T - y \cdot \mathsf{point}^2] \cdot \Gamma + \mathsf{share}_v$
5 : $\mathsf{base}_\alpha \leftarrow \mathsf{share}_\alpha - \mathsf{mid}_\alpha \cdot \mathsf{point}$
6 : **return** base_α

4.7 Key generation

Algorithm 6 RYDE.KeyGen

KeyGen(λ)

1 : $\mathsf{seed}_{\mathsf{sk}} \leftarrow \{0,1\}^\lambda$
2 : $\mathsf{seed}_{\mathsf{pk}} \leftarrow \{0,1\}^\lambda$
3 : $s', C \leftarrow \mathrm{ExpandSecret}(\mathsf{seed}_{\mathsf{sk}})$ $\quad \triangleright\ s' \in \mathbb{F}_{q^m}^{r-1}, C \in \mathbb{F}_q^{r \times (n-r)}$
4 : $H \leftarrow \mathrm{ExpandMatrixH}(\mathsf{seed}_{\mathsf{pk}})$ $\quad \triangleright\ H \in \mathbb{F}_{q^m}^{(n-k) \times k}$
5 : $y \leftarrow (I_{n-k} \parallel H) \cdot (1 \parallel s') \cdot (I_r \parallel C)$ $\quad \triangleright\ \boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$
6 : $pk \leftarrow (\mathsf{seed}_{\mathsf{pk}}, y)$
7 : $sk \leftarrow (\mathsf{seed}_{\mathsf{sk}}, \mathsf{seed}_{\mathsf{pk}})$
8 : **return** (pk, sk)

4.8 Sign

Algorithm 7 RYDE.Sign

Sign(sk, msg)

// *Step 0: Initialization*
$s', C \leftarrow \text{ExpandSecret}(\text{seed}_{\text{sk}}),\ H \leftarrow \text{ExpandMatrixH}(\text{seed}_{\text{pk}})$
$y \leftarrow (I_{n-k} \parallel H) \cdot (1 \parallel s') \cdot (I_r \parallel C),\ pk \leftarrow (\text{seed}_{\text{pk}}, y)$
$\text{salt} \leftarrow \{0,1\}^{2\lambda}, \text{rseed} \leftarrow \{0,1\}^{\lambda}$
// *Step 1: Build and commit to witness polynomials*
$\text{tree} \leftarrow \text{Tree.PRG}(\text{salt}, \text{rseed})$
for $e \in [1:\tau]$
  $\text{acc}[e] \leftarrow (0,0,0),\ \text{base}[e] \leftarrow (0,0,0)$
  for $i \in [1:N_e]$
    $\text{seeds}[e][i] \leftarrow \text{seeds}[N-1+\psi(e,i)]$
    $\text{com}[e][i] \leftarrow \text{Commit}(\text{salt}, e, i, \text{seeds}[e][i])$
    $\overline{s'}, \overline{C}, \overline{v} \leftarrow \text{ExpandShare}(\text{seeds}[e][i], \text{salt})$
    $\text{acc}[e] \leftarrow \text{acc}[e] + (\overline{s'}, \overline{C}, \overline{v})$
    $\text{base}[e] \leftarrow \text{base}[e] + (\phi(i) \cdot \overline{s'}, \phi(i) \cdot \overline{C}, \phi(i) \cdot \overline{v})$
  $(\text{acc}_{s'}, \text{acc}_C, \text{acc}_v) \leftarrow \text{acc}[e]$
  $(\text{aux}_{s'}[e], \text{aux}_C[e], v[e]) \leftarrow (s' - \text{acc}_{s'}, C - \text{acc}_C, \text{acc}_v)$
$h_1 \leftarrow \text{Hash}_1(\text{salt}, \text{com}[*][*], \{\text{aux}_{s'}[e], \text{aux}_C[e]\}_{e\in[1:\tau]})$
// *Step 2: Compute the polynomial proof* $P_\alpha(X)$
$\Gamma \leftarrow \text{ExpandChallenge1}(h_1)$
for $e \in [1:\tau]$
  $(\text{mid}_\alpha[e], \text{base}_\alpha[e]) \leftarrow \text{ComputePolynomialProof}(\text{base}[e], \boldsymbol{v}[e], \boldsymbol{s'}, \boldsymbol{C}, \boldsymbol{\Gamma}, \boldsymbol{H})$
// *Step 3: Open random evaluations*
$\text{ctr} \leftarrow 0$
$h_2 \leftarrow \text{Hash}_2(\text{Hash}_0(msg), pk, \text{salt}, h_1, \{\text{base}_\alpha[e], \text{mid}_\alpha[e]\}_{e\in[1:\tau]})$
`retry:`
$\{i^*[e]\}_{e\in[1:\tau]}, v_{grinding} \leftarrow \text{ExpandChallenge2}(h_2, \text{ctr})$
$\text{path} \leftarrow \text{Tree.GetSiblingPath}(\text{tree}, \{\psi(e, i^*[e])\}_{e\in[1:\tau]})$
if path has at least $T_{open}+1$ nodes or $v_{grinding} \in \{0,1\}^w$ is different from zero
  $\text{ctr} \leftarrow \text{ctr} + 1$
goto `retry`
$\sigma \leftarrow \big(\text{salt} \parallel \text{ctr} \parallel h_2 \parallel \text{path} \parallel \{\text{aux}_{s'}[e], \text{aux}_C[e], \text{mid}_\alpha[e], \text{com}[e][i^*[e]]\}_{e\in[1:\tau]}\big)$
return σ

4.9 Verify

Algorithm 8 RYDE.Verify

Verify($\mathsf{pk}, \sigma, \mathsf{msg}$)

1 : // *Step 0: Initialization*
2 : $(\mathsf{salt} \parallel \mathsf{ctr} \parallel h_2 \parallel \mathsf{path} \parallel \{\mathsf{aux}_s[e], \mathsf{aux}_C[e], \mathsf{mid}_\alpha[e], \mathsf{com}^*[e]\}_{e\in[1:\tau]}) \leftarrow \sigma$
3 : $H \leftarrow \mathrm{ExpandMatrixH}(\mathsf{seed}_{\mathsf{pk}})$
4 : $\{i^*[e]\}_{e\in[1:\tau]}, v_{grinding} \leftarrow \mathsf{ExpandChallenge2}(h_2, \mathsf{ctr})$
5 :
6 : // *Step 1: Computing opened evaluations*
7 : $\mathsf{seeds} \leftarrow \mathsf{Tree.GetSeedsFromPath}(\{\psi(e, i^*[e])\}_{e\in[1:\tau]}, \mathsf{salt})$
8 : **for** $e \in [1:\tau]$
9 : $\quad \mathsf{share}[e] \leftarrow (0,0,0)$
10 : $\quad$ **for** $i \in [1:N_e]$
11 : $\quad\quad$ **if** $i = i^*[e]$
12 : $\quad\quad\quad \mathsf{com}[e][i] \leftarrow \mathsf{com}^*[e]$
13 : $\quad\quad$ **else**
14 : $\quad\quad\quad \mathsf{com}[e][i] \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seeds}[e][i])$
15 : $\quad\quad\quad \overline{s'}, \overline{C}, \overline{v} \leftarrow \mathsf{ExpandShare}(\mathsf{seeds}[e][i], \mathsf{salt})$
16 : $\quad\quad\quad \mathsf{scalar} \leftarrow \phi(i^*[e]) - \phi(i)$
17 : $\quad\quad\quad \mathsf{share}[e] \leftarrow \mathsf{share}[e] + (\mathsf{scalar} \cdot \overline{s'}, \mathsf{scalar} \cdot \overline{C}, \mathsf{scalar} \cdot \overline{v})$
18 : $\quad\quad\quad \mathsf{share}[e] \leftarrow \mathsf{share}[e] + (\phi(i^*[e]) \cdot \mathsf{aux}_{s'}[e], \phi(i^*[e]) \cdot \mathsf{aux}_C[e], 0)$
19 : $h_1 \leftarrow \mathsf{Hash}_1(\mathsf{salt}, \mathsf{com}[*][*], \{\mathsf{aux}_{s'}[e], \mathsf{aux}_C[e]\}_{e\in[1:\tau]})$
20 :
21 : // *Step 2: Recompute the polynomial proof* $P_\alpha(X)$
22 : $\Gamma \leftarrow \mathsf{ExpandChallenge1}(h_1)$
23 : **for** $e \in [1:\tau]$
24 : $\quad \mathsf{point} \leftarrow \phi(i^*[e])$
25 : $\quad \mathsf{base}_\alpha[e] \leftarrow \mathsf{RecomputePolynomialProof}(\mathsf{point}, \mathsf{share}[e], \boldsymbol{\Gamma}, \boldsymbol{H}, \boldsymbol{y}, \mathsf{mid}_\alpha[e])$
26 :
27 : // *Step 3: Verification*
28 : $h'_2 \leftarrow \mathsf{Hash}_2(\mathsf{Hash}_0(msg), pk, \mathsf{salt}, h_1, \{\mathsf{base}_\alpha[e], \mathsf{mid}_\alpha[e]\}_{e\in[1:\tau]})$
29 : **return** $(h_2 \stackrel{?}{=} h'_2)$ **and** $(v_{grinding} \stackrel{?}{=} \mathbf{0}_w)$

5 Parameter Sets

We provide several parameter sets using the nomenclature RYDE-X-y where $\mathsf{X} \in \{1, 3, 5\}$ denotes the security level and $\mathsf{y} \in \{\mathsf{short}, \mathsf{fast}\}$ refers to size / performance trade-off considered for the parameter set.

5.1 RSD parameters

RSD parameters used in RYDE are given in Tables 4. The security of RYDE against the attacks on the Rank Syndrome Decoding problem are estimated by using the CryptographicEstimators V2.0.0 [2] [EVZB24], which considers all the classical attacks described in Section 8. In our estimations, we take ω, the linear algebra constant, to be 2.

Instance	q	m	n	k	r
RYDE-1	2	53	53	45	4
RYDE-3	2	61	61	51	5
RYDE-5	2	67	67	55	6

Table 4. RSD parameters used in RYDE

5.2 Protocol parameters

The protocol related parameters used in RYDE are given in Table 5.

Instance	ρ	τ	N	T_{open}	w
RYDE-1-Short	3	11	2^{12}	116	7
RYDE-1-Fast	3	17	2^{8}	118	9
RYDE-3-Short	4	17	2^{12}	174	5
RYDE-3-Fast	4	26	2^{8}	184	10
RYDE-5-Short	4	23	2^{12}	232	3
RYDE-5-Fast	4	36	2^{8}	244	4

Table 5. MPC parameters used in RYDE

5.3 Key and Signature sizes

Table 6 presents the public key, secret key, and signature sizes of RYDE. The public key pk has $(\lambda + (n-k) \cdot \log_2 q)$ bits, while the secret key sk has λ bits.

[2] Code available at https://github.com/Crypto-TII/CryptographicEstimators.

The bit-length of a RYDE signature σ is given by

$$|\sigma| = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{64}_{\mathsf{ctr}} + \underbrace{2\lambda}_{h_2} + \underbrace{\lambda \cdot T_{\text{open}}}_{\mathsf{path}} + \tau \cdot (\underbrace{2\lambda}_{\mathsf{com}_{i^*}} + \underbrace{(r - 1 + \rho) \cdot \log_2 q}_{(\mathsf{aux}_{s'}, \mathsf{mid}_{\alpha})} + \underbrace{r \cdot (n - r)}_{\mathsf{aux}_C}).$$

Instance	sk	pk	σ
RYDE-1-Short	32 B	69 B	2 988 B
RYDE-1-Fast	32 B	69 B	3 597 B
RYDE-3-Short	48 B	101 B	6 728 B
RYDE-3-Fast	48 B	101 B	8 264 B
RYDE-5-Short	64 B	133 B	11 819 B
RYDE-5-Fast	64 B	133 B	14 609 B

Table 6. Keys and signature sizes of RYDE.

6 Implementation and Performance Analysis

This section provides performance measures of our implementations of RYDE.

Benchmark platform. The benchmarks were performed on a machine running Ubuntu Server 22.04.5 LTS, equipped with an Intel 13th-generation Intel (R) Core(TM) i9-13900K CPU running at 3000MHz and 64GB of RAM. All the experiments were performed with Hyper-Threading, Turbo Boost, and SpeedStep features disabled. The scheme has been compiled with GCC compiler (version 11.4.0) and uses the XKCP library.

The results of each parameter set were obtained by computing the mean from 25 random instances. To minimize biases from background tasks running on the benchmark platform, each instance has been repeated 25 times and averaged. Additionally, all the parameter sets run without increasing the stack memory, except RYDE-5-Short, which requires increasing the stack memory to 9728 kilobytes (e.g., we increase the stack memory by running `ulimit -s 9728`).

Constant time. The provided implementations have been implemented in a constant time way whenever relevant, and as such, the running time is expected to not leak any information concerning sensitive data. Additionally, Valgrind (version 3.18.1) and LibVEX were used to check that the implementation did not have memory leaks.

Remark on the instantiation of RYDE. The performance profile of RYDE (and more generally any MPCitH based schemes) is highly dependent on the performances of the underlying symmetric primitives. In RYDE, PRG are instantiated using either AES or SHAKE while hash functions are instantiated using SHA3. Hereafter, we provide benchmarks with two different instantiations for commitment schemes namely one based on Rijndael and one based on SHA3. One should note that different instantiation choices lead to significant differences in performances. We may provide additional benchmarks with different symmetric primitive choices in the future.

6.1 Reference Implementation

The performance concerning the reference implementation on the aforementioned benchmark platform is described in Tables 7 and 8 respectively. One should note that our reference implementation is only provided to help understanding the scheme and as such is not representative of the performance that the scheme can achieve, we defer the reader to Tables 9 and 10 for the performances of our optimized implementation.

The following optimization flags have been used during compilation:

– Concerning the C code: `-O3 -flto`.

– Concerning the ASM code (required in the XKCP library): `-x assembler-with-cpp -Wa,-defsym,old_gas_syntax=1 -Wa,-defsym,no_plt=1`.

Instance	KeyGen	Sign	Verify
RYDE-1-Short	126.7 K	20.8 B	20.8 B
RYDE-1-Fast	124.0 K	2.0 B	2.0 B
RYDE-3-Short	196.2 K	89.5 B	89.4 B
RYDE-3-Fast	199.2 K	8.6 B	8.5 B
RYDE-5-Short	491.7 K	109.4 B	109.1 B
RYDE-5-Fast	492.3 K	10.7 B	10.6 B

Table 7. Performances of RYDE reference implementation in Thousand (K) and Billion (B) of CPU cycles using Rijndael-based commitment schemes.

Instance	KeyGen	Sign	Verify
RYDE-1-Short	126.1 K	16.1 B	16.1 B
RYDE-1-Fast	123.7 K	1.6 B	1.5 B
RYDE-4-Short	196.4 K	69.0 B	68.9 B
RYDE-3-Fast	199.4 K	6.6 B	6.5 B
RYDE-5-Short	490.0 K	81.7 B	81.4 B
RYDE-5-Fast	490.7 K	8.0 B	7.9 B

Table 8. Performances of RYDE reference implementation in Thousand (K) and Billion (B) of CPU cycles using SHA3-based commitment schemes.

6.2 Optimized Implementation

The performance concerning the optimized implementation on the aforementioned benchmark platform is described in Tables 9 and 10.

The following optimization flags have been used during compilation:

– Concerning the C code: `-O3 -flto -mpclmul -msse2 -mavx -mavx2 -maes`.
– Concerning the ASM code (required in the XKCP library): `-x assembler-with-cpp -Wa,-defsym,old_gas_syntax=1 -Wa,-defsym,no_plt=1`.

Instance	KeyGen	Sign	Verify
RYDE-1-Short	33.7 K	47.7 M	44.6 M
RYDE-1-Fast	33.9 K	4.6 M	4.4 M
RYDE-3-Short	61.0 K	246.0 M	229.1 M
RYDE-3-Fast	61.3 K	22.7 M	22.1 M
RYDE-5-Short	75.9 K	525.0 M	399.2 M
RYDE-5-Fast	75.8 K	42.8 M	40.0 M

Table 9. Performances of RYDE optimized implementation in Thousand (K) and Million (M) of CPU cycles using Rijndael-based commitment schemes.

Instance	KeyGen	Sign	Verify
RYDE-1-Short	33.5 K	99.3 M	96.3 M
RYDE-1-Fast	33.7 K	9.7 M	9.4 M
RYDE-3-Short	61.2 K	286.4 M	277.5 M
RYDE-3-Fast	61.0 K	26.7 M	26.9 M
RYDE-5-Short	76.0 K	685.6 M	558.2 M
RYDE-5-Fast	75.7 K	59.1 M	55.7 M

Table 10. Performances of RYDE optimized implementation in Thousand (K) and Million (M) of CPU cycles using SHA3-based commitment schemes.

6.3 Known Answer Test Values

Known Answer Test (KAT) values have been generated using the script provided by the NIST and can be retrieved in the KATs/ folder. Both reference and optimized implementations generate the same KATs. In addition, examples with intermediate values have also been provided in these folders. The intermediate values correspond with one execution calling the NIST-provided randombytes function seeded with zero. One can generate the test files as mentioned above using the kat and verbose modes of the implementation, respectively. The procedure is detailed in the technical documentation (README file).

7 Security Analysis

7.1 Security Proof

In this section, we provide a security proof for the RYDE scheme. One should note that this proof relies on generic PRGs, Hash functions and commitment schemes and as such does not encompass the specific choices made in order to instantiate these primitives while implementing the scheme.

Theorem 1. *Let the PRG used be (t, ϵ_{PRG})-secure, and ϵ_{RSD} the advantage an adversary has over the specialized Rank Syndrome Decoding problem (RSD_s). Consider $\mathsf{Hash}_0, \mathsf{Hash}_1, \mathsf{Hash}_2, \mathsf{Hash}_3$ behave as random oracles, with an output of 2λ bits. Then, if an adversary makes q_i queries to Hash_i and q_S queries to the signing oracle, the probability for him to produce a forgery for the RYDE signature scheme is given by:*

$$\begin{aligned}\mathsf{Pr}[\mathsf{Forge}] \leq & \frac{4 \cdot (q' + \tau \cdot N \cdot q_S)^2}{2 \cdot 2^{2\lambda}} + \frac{q_S \cdot (q_S + 4q')}{2^{2\lambda}} \\ & + q_S \cdot \tau \cdot \epsilon_{PRG} + q' \cdot 2^{-w} \cdot \left(\frac{2}{N}\right)^{\tau} + q' \cdot \tau \cdot \frac{1}{q^{m \cdot \rho}} + \epsilon_{RSDs}\end{aligned}$$

where $q' = \max(q_1, q_2, q_3)$ and τ is the number of repetitions of the signature.

Proof. In this proof, we will adopt a game hopping strategy in order to find the upper bound. The first game will be the access to the standard signing oracle by the adversary $\mathcal{A}$. We will then game hop in order to eliminate the cases where collisions happen, and, through some other games, we will manage to find an upper bound. We note $\mathsf{Pr}_i[\mathsf{Forge}]$ the probability of forgery when considering game i. The aim of the proof is to find an upper bound on $\mathsf{Pr}_1[\mathsf{Forge}]$.

- **Game 1.** This is the interaction between $\mathcal{A}$ and the real signature scheme. KeyGen generates $(\boldsymbol{H}, \boldsymbol{y}, \boldsymbol{s'}, \boldsymbol{C'})$ and $\mathcal{A}$ receives $(\boldsymbol{H}, \boldsymbol{y})$. $\mathcal{A}$ can make queries to each Hash_i independently, and can make signing queries. At the end of the attack, $\mathcal{A}$ outputs a message/signature pair, (m, σ). The event Forge happens when the message output by $\mathcal{A}$ was not previously used in a query to the signing oracle.
- **Game 2.** In this game, we add a condition to the success of the attacker. The condition we add is that if there is a collision between outputs of Hash_0, or Hash_1, or Hash_2, or Hash_3, then, the forgery is not valid. The first step is to look at the number of times every Hash_i is called when calling the signing oracle. The signing oracle contains one call to Hash_0, one call to Hash_1 and Hash_2, and $\tau \cdot N$ to Hash_3. The number of queries to Hash_0, or Hash_1, to Hash_2, or to Hash_3 is then bounded from above by $q' + \tau \cdot N \cdot q_S$, where q_i is the number of queries made by $\mathcal{A}$ to Hash_i, $q' = \max\{q_0, q_1, q_2, q_3\}$, q_S is the number of queries to the signing oracle. We thus have:

$$|\mathsf{Pr}_1[\mathsf{Forge}] - \mathsf{Pr}_2[\mathsf{Forge}]| \leq \frac{4(q' + \tau \cdot N \cdot q_S)^2}{2 \cdot 2^{2\lambda}}$$

– **Game 3.** The attacker now fails if the inputs to any of the Hash_i has already appeared in a previous query. If that happens, this means that at least the salt used was the same (we emphasize on *at least*). We have one salt sampled every time a query is made to the signing oracle, and it can collide each time with: a previous salt, or any of the queries to the Hash_i. This means, we can bound this with:

$$|\mathsf{Pr}_2[\mathsf{Forge}] - \mathsf{Pr}_3[\mathsf{Forge}]| \leq \frac{q_S \cdot (q_S + q_0 + q_1 + q_2 + q_3)}{2^{2\lambda}} \quad \leq \frac{q_S \cdot (q_S + 4 \cdot q')}{2^{2\lambda}}$$

– **Game 4.** When signing a message m, we now replace h_1 and h_2 with uniformly distributed random values. We then compute the challenges $\boldsymbol{\Gamma}$ and $\{i^{*(e)}\}_e$, and the value v_{grinding}, by expanding them. There is a difference with **Game 3** during a signing query, if a query to Hash_1 or Hash_2 was previously made. However, this does not happen as **Game 3** would already abort due to salt collision which means

$$\mathsf{Pr}_4[\mathsf{Forge}] = \mathsf{Pr}_3[\mathsf{Forge}]$$

– **Game 5.** We replace each $\mathsf{com}_{e,i}$, with a uniformly distributed random value. Since $\mathsf{com}_{e,i}$ is expanded using e and i, his then means that, to have a collision on two queries to Hash_3, the same salt must be used, which is already an invalid forgery from the previous games. Thus,

$$\mathsf{Pr}_5[\mathsf{Forge}] = \mathsf{Pr}_4[\mathsf{Forge}]$$

– **Game 6.** We now use the HVZK simulator of the proof of knowledge in order to generate the views of the parties randomly. $\mathcal{A}$ has an advantage of ϵ_{PRG} at most when generating the views (as it is his advantage to distinguish between a random and a true transcript). Hence, the difference with the previous game is given by:

$$|\mathsf{Pr}_6[\mathsf{Forge}] - \mathsf{Pr}_5[\mathsf{Forge}]| \leq \tau \cdot q_S \cdot \epsilon_{PRG}$$

– **Game 7.** Finally, we say that an execution with index (e^*, ctr) of a query $h_2 = \mathsf{Hash}_2\left(\mathsf{Hash}_0(msg), \mathsf{pk}, \mathsf{salt}, h_1, \{\mathsf{base}_\alpha^{(e)}, \mathsf{mid}_\alpha^{(e)}\}_e\right)$ allows to retrieve a correct witness if:

- h_1 is a query to Hash_1, i.e., $h_1 = \mathsf{H}_1(\mathsf{salt}, \{\mathsf{com}_{e,i}\}_{e,i}, \{\mathsf{aux}_{s'}^{(e)}, \mathsf{aux}_{C'}^{(e)}\})$;
- Each $\mathsf{com}_{e,i}$ is a query to Hash_3, i.e., $\mathsf{com}_{e,i} = \mathsf{Hash}_3(\mathsf{salt}, e, i, \mathsf{seed}_{e,i})$;
- The vector and matrix $\boldsymbol{s'}, \boldsymbol{C'}$ defined by $\{\mathsf{state}_i^{(e^*)}\}_{i\in[1,\dots,N]}$ form a correct $\mathsf{RSD_s}$ solution;
- $v_{\text{grinding}} = \mathbf{0} \in \{0,1\}^w$ and $\pi_{\mathsf{BAVC}} \neq \perp$.

In such cases, one is able to retrieve the correct witness from $\{\mathsf{seed}_i^{(e^*)}\}_{i\in[1,\dots,N]}$, and as a consequence is able to solve the specialized Rank Syndrome Decoding instance. This means that

$$\mathsf{Pr}_7[\mathsf{Solve}] \leq \epsilon_{RSDs}$$

Finally, we need to look at the upper bound of $\left|\mathsf{Pr}_7[\mathsf{Forge} \cap \overline{\mathsf{Solve}}]\right|$. Solve does not happen here, meaning that, to have a forgery after a query to Hash_2, $\mathcal{A}$ has no choice but to cheat either on $\boldsymbol{\Gamma}$ or on $\{i^{*(e^*)}\}$. For that, he can:

- Find a vector $\tilde{\boldsymbol{s'}}$ and a matrix $\tilde{\boldsymbol{C'}}$ such that $\left((1 \parallel \tilde{\boldsymbol{s'}}) \cdot \left[\boldsymbol{I}_r \parallel \tilde{\boldsymbol{C'}}\right] \cdot \boldsymbol{H}^\top\right) \neq \boldsymbol{y}$ but such that $\boldsymbol{\Gamma} \cdot \left((1 \parallel \tilde{\boldsymbol{s'}}) \cdot \left[\boldsymbol{I}_r \parallel \tilde{\boldsymbol{C'}}\right] \cdot \boldsymbol{H}^\top\right) = \boldsymbol{y}$, which happens with probability $p = \frac{1}{q^{m \cdot \rho}}$;
- Successfully cheat on the polynomial P_α, which happens with probability $\frac{2}{N}$ because there are 2 roots to this polynomial.

Cheating on the second round must happen on the τ repetitions thus the cheating probability is bounded by $q' \cdot \left(\frac{2}{N}\right)^\tau + q' \cdot \tau \cdot \frac{1}{q^{m \cdot \rho}}$. However, because v_{grinding} must be equal to $\mathbf{0}$, the adversary $\mathcal{A}$ has a success probability 2^{-w} for each iteration (e^*, ctr). This results in a probability to cheat of $2^{-w} q' \cdot \left(\frac{2}{N}\right)^\tau + q' \cdot \tau \cdot \frac{1}{q^{m \cdot \rho}}$. Finally, the adversary must have $\pi_{\mathsf{BAVC}} \neq \perp$. Let θ be the probability that $\pi_{\mathsf{BAVC}} = \perp$ namely the probability that the sibling path exceeds the threshold T_{open}. This reduces the number of possible challenges from N^τ to $(N^\tau) \cdot (1 - \theta)$. Thus, the adversary only has to guess among the $(N^\tau) \cdot (1 - \theta)$ challenges which does not fail, but since the set $\{i^{*(e)}\}_e$ is uniformly sampled, the forgery will fail with probability θ. As a result, the adversary can cheat with probability $\frac{2^\tau}{(N^\tau) \cdot (1-\theta)} \cdot (1 - \theta) = \left(\frac{2}{N}\right)^\tau$.

Computing the sum of the aforementioned upper bounds concludes the proof.

8 Known Attacks

8.1 Generic Attacks against Fiat-Shamir Signatures

It is possible to forge a signature without solving the underlying instance of the MinRank problem. For signature schemes built by applying the Fiat-Shamir transformation on a five-pass identification, Kales and Zaverucha proposed in [KZ20] a forgery achieved by guessing separately the two challenge of the protocol. It results in an additive cost rather than the expected multiplicative cost. The cost associated with forging a transcript that passes the first 5 rounds of the Proof of Knowledge relies on achieving an optimal trade-off between the work needed for passing the first step and the work needed for passing the second step. To achieve the attack, one can find an optimal number of repetitions with the formula:

$$\tau' = \arg \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{P_1} + \left(\frac{1}{P_2} \right)^{\tau - \tau'} \right\}$$

where P_1 and P_2 are the probabilities to pass respectively the first τ challenge τ' times and the second challenge $\tau - \tau'$ times.

In our case, P_1 corresponds to the probability of having a false-positive in the polynomial constraints protocol ρ times, and P_2 corresponds to the probability of cheating on the polynomial P_α (see Section 3.1). Therefore, the KZ attack complexity is given by

$$\text{cost}_{\text{forge}} = \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau - i}} + (\frac{N}{2})^{\tau - \tau'} \right\}$$

where $p = \frac{1}{q^{m \cdot \rho}}$.

8.2 Known Attacks against Rank Syndrome Decoding

Combinatorial attacks. We describe in this section the most efficient combinatorial attacks against Rank-SD problem.

Enumeration of basis. Chabaud and Stern proposed an algorithm in [CS96] which solve the problem by enumerating the possible supports for the vector $\boldsymbol{x}$. For each of them, one must translate the equations in $\mathbb{F}_{q^m}$ to equations in $\mathbb{F}_q$.

If trying all the different possible supports in Hamming metric is inefficient due to their large number (there are $\binom{n}{\omega}$ vectors of weight ω in $\mathbb{F}_2^n$), this is a viable method in rank metric: there are approximately $q^{(m-r)r}$ linear subspaces in $\mathbb{F}_{q^m}$ of dimension r. Moreover, one can always make the assumption that 1 is in the support. There are approximately $q^{(m-r)(r-1)}$ linear subspaces in $\mathbb{F}_{q^m}$ which contains 1.

Hence, one obtains a system that has $(n-k)m$ independent equations and $nr+m$ variables in $\mathbb{F}_q$. Thanks to the Gaussian elimination, this system can be solved with $O((nr+m)^\omega)$ operations in $\mathbb{F}_q$. It is deduced that the complexity of this attack is upper bounded by $O((nr+m)^\omega q^{(m-r)(r-1)})$.

Ourivski-Johansson. The attack [OJ02] first apply a well-known reduction, consisting in adding the syndrome $\boldsymbol{y}$ to the code $\mathcal{C}$, and then exhibits a system of quadratic equations. The aim of this attack is to linearize the equations, which is done after fixing a number of values. This algorithm solves the problem in

$$\mathcal{O}\left((rm)^{\omega} q^{(r-1)(k+1)}\right).$$

GRS algorithm. This method is an adaptation of the information set decoding attack used in Hamming metric. From the syndrome $\boldsymbol{y}$ of length $n-k$, the algorithm consists in guessing a set of $n-k$ coordinates which contains the support of the vector $\boldsymbol{x}$. One obtains $n-k$ equations and $n-k$ unknown coordinates of $\boldsymbol{x}$, which can be recovered by inverting an extracted matrix from $\boldsymbol{H}$ of size $(n-k)\times(n-k)$.

In order to adapt this method to coding theory in rank metric, Gaborit, Ruatta and Schrek proposed in [GRS16] to consider a linear subspace in $\mathbb{F}_{q^m}$ of dimension $r' \geq r$, and hope that it includes the support of the vector $\boldsymbol{x}$. The probability that E of dimension r is included in E' of dimension r' (for $r' \geq r$) is $q^{-(m-r')r}$.

Suppose one knows a subspace E' of dimension r' which contains $E = \mathrm{Supp}(\boldsymbol{x})$. It is possible to recover $\boldsymbol{x}$ by solving a linear system as long as

$$r'n \leq (n-k)m \qquad \text{i.e.} \qquad r' \leq \left\lceil \frac{(n-k)m}{n} \right\rceil$$

Choosing the higher possible value $r = \lceil \frac{(n-k)m}{n} \rceil$, one gets the probability to obtain a suitable subspace E' equal to $q^{-(m-r')r} = q^{-r\lceil \frac{km}{n} \rceil}$. As in the previous attack, one can recover x by executing Gaussian elimination, which can be perform in $O((n-k)^{\omega} m^{\omega})$ operations in $\mathbb{F}_q$. This attack can be achieved with an average complexity: $O((n-k)^{\omega} m^{\omega} q^{r\lceil \frac{km}{n} \rceil})$.

Improved GRS. The idea proposed in [AGHT18] is, instead of considering a linear subspace in $\mathbb{F}_{q^m}$ which contains 1, to choose a completely random subspace E'. If E' contains a subspace of the form αE, with $\alpha \in \mathbb{F}_{q^m}^*$, then one can retrieve the code word as before. At the cost of an increase in the dimension of the code (k becomes $k+1$, since we multiply by an element α which does not belong to the code), we can obtain a new subspace that contains αE. This attack can be achieved with an average complexity: $\mathcal{O}\left((n-k)^{\omega} m^{\omega} q^{(r-1)\lceil \frac{(k+1)m}{n} \rceil}\right)$, where ω is in practice equal to 2.

Guessing-enhanced GRS. More recently, [DEGS24] improved the GRS algorithm for some types of parameters. In a nutshell, their improvement comes from the fact that it is possible to guess only some bits of elements of $\mathbb{F}_{q^m}$, instead of an entire element. The gain comes from the term $\lceil \frac{(k+1)m}{n} \rceil$, which can get reduced enough depending on its value. The complexity of their algorithm is $\mathcal{O}\left(\min_{t \leq mn}\{((n-k)m+t)^{\omega} \cdot \max\left(1, q^{r\lceil \frac{(k+1)m-t}{n} \rceil + t - m}\right)\}\right)$.

Algebraic attacks. Algebraic attacks amount to solve the system of equations $\boldsymbol{H}\boldsymbol{x}^\top = \boldsymbol{y}^\top$ using computer algebra techniques like Gröbner basis. Today, the best algebraic modelings for solving the Rank-SD problem are the MaxMinors modeling [BBB+20,BBC+20] and the Support Minors modeling [BBC+20,BBB+23]. They can both benefits from a classical improvement that is the hybrid approach, that consists in specializing some variables with all possible values and solving the resulting system with less variables, but the same number of equations. In certain cases, the gain in complexity for solving the specialized system compared to the original one superseeds the lost in complexity coming from the exhausive search.

Hybrid methods. As shown in [BBB+23, Section 5], solving a Rank-SD problem of parameters (q, m, n, k, r) amount to solve q^{ar} smaller Rank-SD problems of parameters $(q, m, n-a, k-a, r)$.

MaxMinors Modeling. Let $\boldsymbol{H}, \boldsymbol{x}, \boldsymbol{y}$ be a Rank-SD problem with parameters (q, m, n, k, r), i.e. $\boldsymbol{H} = (\boldsymbol{I_{n-k}} \parallel \boldsymbol{H'}) \in \mathbb{F}_{q^m}^{(n-k)\times n}$ is uniformly sampled, $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$ has weight exactly[3] r, and $\boldsymbol{y}^\top = \boldsymbol{H}\boldsymbol{x}^\top$.

The idea of the MaxMinors modeling from [BBB+20] is the following: let $\boldsymbol{E} = (E_1, \ldots, E_r)$ be a basis of $\mathrm{Supp}(\boldsymbol{x})$, and $\boldsymbol{X} = (x_{i,j}) \in \mathbb{F}_q^{r\times n}$ a matrix representing the coordinates of $\boldsymbol{x}$ in the basis $\boldsymbol{E}$, then $\boldsymbol{x} = (E_1, \ldots, E_r)\boldsymbol{X}$. Considering the extended code C' described in the previous section, and a parity-check $\boldsymbol{H'} \in \mathbb{F}_{q^m}^{(n-k-1)\times n}$ of this code, we have the relation :

$$\boldsymbol{H'}\boldsymbol{x}^\top = 0, \text{ i.e. } (E_1, \ldots, E_r)\boldsymbol{X}\boldsymbol{H'}^\top = 0.$$

This implies that the matrix $\boldsymbol{X}\boldsymbol{H'}^\top$ is not full rank, and that all its maximal minors are equal to zero. By using the Cauchy-Binet formula that expresses the determinant of the product of two matrices $A \in \mathbb{K}^{r\times n}$ and $B \in \mathbb{K}^{n\times r}$ as

$$|AB| = \sum_{T\subset\{1..n\},\#T=r} |A|_{*,T}\,|B|_{T,*}, \tag{5}$$

each of these maximal minors can be expressed as a linear combination of the maximal minors of the matrix $\boldsymbol{X}$. Using these minors as new variables, we obtain a system of $\binom{n-k-1}{r}$ linear equations in $\binom{n}{r}$ variables with coefficients over $\mathbb{F}_{q^m}$, that can be unfolded over $\mathbb{F}_q$ as $m\binom{n-k-1}{r}$ equations in the same number of variables (that are searched over $\mathbb{F}_q$).

It is possible to solve the Rank-SD problem with the MaxMinors modeling as soon as

$$\begin{cases} m\binom{n-k-1}{r} \geq \binom{n}{r} - 1, & \text{if the system has 1 solution,} \\ m\binom{n-k-1}{r} \geq \binom{n}{r}, & \text{if the system has no solution.} \end{cases} \tag{6}$$

[3] We assume the weight r is known. Otherwise, we can try to solve for $r = 1$, $r = 2$, ... until we find the good r.

It is possible to improve slightly the solving by puncturing the code on p positions, as long as

$$m\binom{n-p-k-1}{r} \geq \binom{n-p}{r} - 1. \tag{7}$$

With the largest such p, the linear system has almost the same number of equations than variables. If (6) is not satisfied, then we use the hybrid approach and specialize $a \in \{0..k\}$ columns of $\boldsymbol{X}$. The final complexity of the MaxMinors attack is bounded by

$$O\left(q^{ar}\binom{n-a-p}{r}^{\omega}\right) \tag{8}$$

as $n \to \infty$, where ω is the linear algebra constant.

The Support Minors modeling. The previous modeling is a linearization technique, that only works with a large hybrid parameter a for large r. An alternative method is to rely on the Support Minors modeling, which was initially introduced in [BBC+20] to solve generic MinRank instances, and that uses a new set of variables. It has been adapted to instances coming from $\mathbb{F}_{q^m}$-linear codes in [BBB+23].

Write the received vector $\boldsymbol{v} = -\boldsymbol{mG} + \boldsymbol{x}$ with $-\boldsymbol{mG} \in C$ a codeword, and $w_R(\boldsymbol{x}) = r$. Then, $\boldsymbol{x} = \boldsymbol{v} + \boldsymbol{mG} = (E_1, \ldots, E_r)\boldsymbol{C}$ so that any row $\boldsymbol{r}_i$ of $\boldsymbol{M}(\boldsymbol{v} + \boldsymbol{mG}) \in \mathbb{F}_q^{m \times n}$ is in the row space of $\boldsymbol{C}$. Therefore, all the maximal minors of the matrix $\binom{\boldsymbol{r}_i}{\boldsymbol{C}}$ are equal to 0. This system is described and analyzed in [BBB+23], where it is shown that it contains the previous equations from the MaxMinors modeling.

It is conjectured in [BBB+23], based on a careful theoretical analysis of the system and some experimental heuristics, that it is possible to solve this bilinear system by linear algebra on a matrix with N rows and M columns, as soon as $N \geq M - 1$, with:

$$N = \sum_{i=1}^{k} \binom{n-i}{r}\binom{k+b-1-i}{b-1} - \binom{n-k-1}{r}\binom{k+b-1}{b} \tag{9}$$

$$- (m-1)\sum_{i=1}^{b}(-1)^{i+1}\binom{k+b-i-1}{b-i}\binom{n-k-1}{r+i}. \tag{10}$$

$$M = \binom{k+b-1}{b}\left(\binom{n}{r} - m\binom{n-k-1}{r}\right). \tag{11}$$

In this case, the final cost in $\mathbb{F}_q$ operations is given by

$$\mathcal{O}\left(m^2 N M^{\omega-1}\right),$$

where ω is the linear algebra constant and where the m^2 factor comes from expressing each $\mathbb{F}_{q^m}$ operation involved in terms of $\mathbb{F}_q$ operations. As previously, it is possible to use an hybrid approach to reach the constraint $N \geq M - 1$, or to puncture the code to get N and M of the same value.

9 Advantages and Limitations

RYDE being a digital signature scheme based on a zero-knowledge proof of knowledge, the scheme benefits from a number of advantages, but possesses a few limitations as well.

9.1 Advantages

Difficulty of the underlying problem. The security of the signature scheme relies on the genuine problem of decoding in rank metric Rank-SD, and there exists a probabilistic reduction from a generic NP-complete problem to the Rank-SD problem [GZ14]. The Rank-SD problem has been studied for many years. In particular the security of the problem corresponds to parameters on the Rank-Gilbert-Varshamov bound, the hardest area for parameters in which recent algebraic attacks [BBB+23] behave similarly to classical combinatorial attacks.

Size of public key and signature. RYDE offers competitive signature sizes using very small public keys, which yields a competitive signature + public key size. For NIST security level I, the sum of the signature and public key sizes of RYDE gives 3.0 kB, which are both smaller than the post-quantum NIST standards ML-DSA (Dillitium) and SLH-DSA (SPHINCS+) with 3.7 kB and 7.8 kB respectively.

No cyclic structure of the underlying problem. Our security is based on a problem which does not rely on cyclic structure for which the quantum security is not fully known.

Resilience against Rank-SD attacks. The size of the signature is composed of two parts: a part related to the MPC that only depends on the security level (seeds, hashes) and a part related to the parameters of the chosen RSD instance. Hence, increasing the size of the problem parameters has a limited impact on the total size of the signature. For example, the RYDE-1-Short instance features a signature size of 3.0 kB using the parameters $(q, m, n, k, r) = (2, 53, 53, 45, 4)$ for NIST security level 1. Choosing the parameters $(q, m, n, k, r) = (2, 61, 61, 51, 5)$ would reach more than 192 bits of security for the underlying problem and result in a signature size of 3.2 kB for NIST security level 1. Thus, if one discovers an efficient attack against the Rank-SD problem that forces us to increase the problem parameters, only the problem parameter part will be impacted and the overall effect on the signature length will be limited.

Size of the public matrix. In our case, the public matrix can be generated from a random seed. Now independently of this, the properties of rank metric make the size of the public matrix very small (and this without additional cyclic structure), for instance of order 1 kB for the first level of parameters: it could be an advantage if one were to choose not to generate the public matrix from a seed.

9.2 Limitations

Growth rate of the signature size. The signature size almost doubles when increasing the security level. This comes from the fact that both the Rank Syndrome Decoding instance and the number of repetitions need to increase linearly, since both the Fiat-Shamir transform and the $\mathsf{RSD}_{\mathrm{s}}$ instance need to reach much higher bit security.

Efficiency. The general TCitH framework (as the other MPCitH frameworks) involves the generation of lot of pseudorandom objects, which makes RYDE slower than the NIST post-quantum standard ML-DSA. Besides this, in general, the efficiency of RYDE is competitive when compared with other post-quantum signature schemes.

Low-cost devices and embedded systems. RYDE might be particularly heavy for low-cost devices such as smart cards or embedded systems, although it has the potential to perform well on hardware as being highly parallelizable.

A Variant using the VOLEitH Framework

In this section, we provide key and signature sizes for RYDE-v, a variant of RYDE using the VOLEitH framework as described in Section 3.1.

Instance	sk size	pk size	σ size
RYDE-v-1-Short	32 B	69 B	2 871 B
RYDE-v-1-Fast	32 B	69 B	3 454 B
RYDE-v-3-Short	48 B	101 B	6 528 B
RYDE-v-3-Fast	48 B	101 B	7 836 B
RYDE-v-5-Short	64 B	133 B	11 788 B
RYDE-v-5-Fast	64 B	133 B	13 900 B

Table 11. Keys and signature sizes of RYDE-v (VOLE variant)

References

AGHT18. Nicolas Aragon, Philippe Gaborit, Adrien Hauteville, and Jean-Pierre Tillich. A New Algorithm for Solving the Rank Syndrome Decoding Problem. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2421–2425, 2018.

BBB+20. Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich. An algebraic attack on rank metric code-based cryptosystems. In *Advances in Cryptology – EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part III*, page 64–93, Berlin, Heidelberg, 2020. Springer-Verlag.

BBB+23. Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, and Jean-Pierre Tillich. Revisiting algebraic attacks on minrank and on the rank decoding problem. *Designs, Codes and Cryptography*, 91:1–37, 07 2023.

BBC+20. Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier Verbel. Improvements of Algebraic Attacks for Solving the Rank Decoding and MinRank Problems. In *Advances in Cryptology – ASIACRYPT 2020*, pages 507–536. Springer International Publishing, 2020.

BBD+23. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023.

BBM+25. Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One tree to rule them all: Optimizing ggm trees and owfs for post-quantum signatures. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 463–493, Singapore, 2025. Springer Nature Singapore.

BFG+24. Loïc Bidoux, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, and Matthieu Rivain. Dual support decomposition in the head: Shorter signatures from rank SD and MinRank. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 38–69, Singapore, 2024. Springer Nature Singapore.

CDI05. Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, Berlin, Heidelberg, February 2005.

CS96. Florent Chabaud and Jacques Stern. The Cryptographic Security of the Syndrome Decoding Problem for Rank Distance Codes. In *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, volume 1163 of *Lecture Notes in Computer Science*, pages 368–381. Springer, 1996.

DEGS24. Giuseppe D'Alconzo, Andre Esser, Andrea Gangemi, and Carlo Sanna. Sneaking up the ranks: Partial key exposure attacks on rank-based schemes. Cryptology ePrint Archive, Paper 2024/2070, 2024.

EVZB24. Andre Esser, Javier A. Verbel, Floyd Zweydinger, and Emanuele Bellini. SoK: CryptographicEstimators - a software library for cryptographic hardness estimation. In Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas, editors, *ASIACCS 24*. ACM Press, July 2024.

Fen24. Thibauld. Feneuil. The Polynomial-IOP Vision of the Latest MPCitH Frameworks for Signature Schemes. Post-Quantum Algebraic Cryptography - Workshop 2, Institut Henri Poincaré, Paris, France, 2024.

FR23a. Thibauld Feneuil and Matthieu Rivain. Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments. Cryptology ePrint Archive, Report 2023/1573, 2023.

FR23b. Thibauld Feneuil and Matthieu Rivain. Threshold linear secret sharing to the rescue of MPC-in-the-head. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 441–473. Springer, Singapore, December 2023.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

GRS16. Philippe Gaborit, Olivier Ruatta, and Julien Schrek. On the complexity of the rank syndrome decoding problem. *IEEE Transactions on Information Theory*, 62(2):1006–1019, 2016.

GZ14. Philippe Gaborit and Gilles Zémor. On the hardness of the decoding and the minimum distance problems for rank codes. *IEEE Transactions on Information Theory*, PP, 04 2014.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

ISN89. Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

KZ20. Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 3–22. Springer, Cham, December 2020.

OJ02. A. V. Ourivski and T. Johansson. New Technique for Decoding Codes in the Rank Metric and Its Cryptography Applications. *Probl. Inf. Transm.*, 38(3):237–246, jul 2002.